2004

Comparative genomics: multiple genome rearrangement and efficient algorithm development

Shiquan Wu
Iowa State University

Comparative genomics: Multiple genome rearrangement and efficient algorithm development

by

Shiquan Wu

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Bioinformatics and Computational Biology

Program of Study Committee:
Xun Gu, Co-major Professor
Zhijun Wu, Co-major Professor
Xiaoqiu Huang
Stephen Willson
Karin Dorman

Iowa State University

Ames, Iowa

2004

Graduate College
Iowa State University

This is to certify that the doctoral dissertation of

Shiquan Wu

has met the dissertation requirements of Iowa State University

Signature was redacted for privacy.

Co-major Professor

Signature was redacted for privacy.

Co-major Professor

Signature was redacted for privacy.

For the Major Program

TABLE OF CONTENTS

ABSTRACT

Multiple genome rearrangement by signed reversal is discussed: For a collection of genomes represented by signed permutations, reconstruct their evolutionary history by using signed reversals, i.e. find a tree where the given genomes are assigned to leaf nodes and ancestral genomes (i.e. signed permutations) are hypothesized at internal nodes such that the total reversal distance summed over all edges of the tree is minimized. It is equivalent to finding an optimal Steiner tree that connects the given genomes by signed reversal paths. The key for the problem is to reconstruct all optimal ancestral genomes or Steiner nodes.

The problem is NP-hard and can only be solved by efficient approximation algorithms. Various algorithms and programs have been designed to solve the problem, such as *BPAnalysis, GRAPPA, grid search algorithm, MGR greedy split algorithm* (Chapter 1). However, they may have expensive computational costs or low inference accuracy. In this thesis, several new algorithms are developed, including *nearest path search algorithm* (Chapter 2), *neighbor-perturbing algorithm* (Chapter 3), *branch-and-bound algorithm* (Chapter 3), *perturbing-improving algorithm* (Chapter 4), and *partitioning algorithm* (Chapter 5). With theoretical proofs, computer simulations, and biological applications, these algorithms are shown to be $2-$approximation algorithms and more efficient than the existing algorithms.

CHAPTER 1. OVERVIEW

In this chapter, we give an overview of the whole dissertation, including the problem we addressed, why the problem is important, how the problem can be solved, results obtained, and possible future work on the problem.

1.1 Problem and motivation

The problem we discuss is multiple genome rearrangement by signed reversal. We motivate the problem in the context of evolutionary history reconstruction. The motivation consists of two parts: The biological motivation and the computational motivation.

1.1 Biological motivation

Ohno's law [16] states that while the gene content of X chromosome has hardly changed throughout 125 million years of mammalian evolution, the order of genes along the chromosomes has been disrupted several times. During evolution, inter- and intra-chromosomal exchange of DNA fragments disrupted the order of genes in the genome. Focusing only on genomic rearrangement, it is interesting to infer a phylogenetic tree showing the historical rearrangements that produced the genomes present today. Consider the evolution of a collection of organisms, we have multiple genome rearrangement, i.e. find a phylogenetic tree showing how genomes evolve from a common ancestor based on gene orders.

1.2 Computational motivation

Computational (or comparative) genomics is one of the most important area in bioinformatics and computational biology. One focus is phylogenetic tree inference. It is interesting and challenging to reconstruct an accurate evolutionary history for a collection of organisms with the least computational cost. Various methods have been developed for phylogenetic tree inference:

(1) **Sequence alignment method** finds all pairwise distances of sequences by sequence comparisons. The phylogenetic tree is then reconstructed from the matrix of the pairwise distances by various algorithms, e.g. neighbor-joining algorithm, UPGMA (Unweighted Pair Group Method using Arithmetic averages), and other algorithms [6, 12, 14, 23].

(2) **Parsimony method** infers a phylogenetic tree by minimizing the total number of evolutionary steps required to explain a given set of data, or by minimizing the total tree length assuming evolution on a bifurcating tree shape. Technically, it computes the minimum number of evolutionary events required to explain the data along all possible topologies and then chooses the tree minimizing the number of evolutionary events.

(3) **Maximum likelihood method** infers the phylogeny of genomes by evaluating a hypothesis of an evolutionary history based on the transition probability of nucleotide substitutions and a hypothesized history would give rise to the observed data set. A higher probability phylogeny is assumed more possible reach the observed state than a lower probability phylogeny. The method searches for the tree with the highest probability or likelihood [6, 7, 9, 10, 11, 12].

All three popular methods may generate incorrect phylogenies, particularly when organisms are very similar. The computational cost is expensive for multiple sequence alignments. In order to improve the accuracy of inference, alternative methods based on comparing gene contents have been proposed.

(4) **Kolmogorov or Lempel-Ziv complexity method** infers phylogenies by comparing gene contents or particular features, tandem repeats, and certain patterns occurring or not occurring in biological sequences [17, 20, 21, 22]. This method improves both the accuracy and the cost of phylogeny inference.

However, all four preceding methods deal with only local mutations. Only individual genes and local comparisons are discussed. Much information has not been taken into account. These methods may give rise to incorrect phylogenetic trees because the actual evolutionary process of any organism contains not only local mutation (e.g. insertion, deletion, and substitution), but also non-local mutations (e.g. reversal, translocation, fission, fusion, etc). Therefore, the entire genome and the orders of genes should be taken into account for inferring phylogenies. For this purpose, the following distance-based method is initiated.

(5) **Evolutionary edit distance method** discusses how to transform the gene order of one genome into another by both local and non-local mutations [18, 19, 24, 26]. This originates multiple genome rearrangement.

Multiple genome rearrangement is extensively discussed under various types of mutations (evolutionary events), including both local mutations (such as insertion, deletion, and substitution) and non-local mutations (such as reversal, translocation, recombination, fission, fusion, etc.)[18, 25]. The more types of mutations are allowed, the more difficult the problem becomes. The general problem is NP-hard [4, 5], so we discuss the problem for various cases. The purpose is to develop efficient approximation algorithms and to find special cases that are polynomial time solvable. Multiple genome rearrangement by signed reversal restricts the discussion to pure signed reversal and becomes an interesting case. It is described as follows:

Multiple genome rearrangement by signed reversal [3, 18, 27, 28] For a given collection of genomes represented by signed permutations, reconstruct their evolutionary

history under signed reversals, i.e. find a bifurcating tree where sampled genomes are assigned to leaf nodes and ancestral genomes (i.e. signed permutations) are hypothesized at internal nodes such that the total reversal distance summed over all edges of the tree is minimized. The problem is equivalent to finding an optimal Steiner tree that connects the genomes by signed reversal paths. To solve the problem means to find the optimal Steiner nodes (i.e. ancestral genomes) of the optimal Steiner tree.

Earlier, a closely related problem has been discussed: Generate a collection of permutations (genomes) $q_j (1 \leq j \leq n)$ from a common ancestral genome (e.g. the identity permutation $p = 12 \cdots |X|$) in the minimum number of signed reversals [13, 27] (X denotes the set of genes).

Another form of the problem is described as follows:

Multiple Genome Rearrangement By Signed Reversal (MGRBSR Problem) Given a collection of permutations $G = \{g_1, g_2, \cdots, g_m\}$, we generate G from some $p \in G$ in the minimum number of signed reversals, i.e. find an optimal Steiner tree to connect the genomes by reversal paths [28].

In various models of genome rearrangement, a genome is defined as a signed permutation. An optimal Steiner node is equivalent to an optimal ancestral genome. And evolutionary history, phylogenetic tree, and optimal Steiner tree are equivalent.

1.2 Previous algorithms

Multiple genome rearrangement is NP-hard [4, 5] and the problem can only be solved by efficient approximation algorithms. Various kinds of algorithms and programs have been developed to solve the problem. We list here several of them that are closely connected to our discussion: (1) the breakpoint analysis [2, 18, 26], (2) the grid search algorithm [25], and (3) the MGR greedy split algorithm[3].

(1) Breakpoint analysis The breakpoint distance between two genomes is defined as the number of consecutive pairs of genes that are adjacent in one genome but not in the other[26]. Breakpoint analysis is one of the earliest methods for genome rearrangement problem based on gene orders. There are several efficient approximation algorithms and programs for the problems, such as GRAPPA [1, 15] and BPAnalysis [2, 18, 26]. However, breakpoint analysis gives many approximation solutions that do not have a clear biological meaning. It could not find a more biologically accurate rearrangement distance [3].

(2) Grid search algorithm [25] deals with a multiple genome rearrangement under signed reversals and transpositions. For three genomes, it constructs a grid by connecting each pair of the three genomes with a shortest path and joining the other genome with each intermediate node in the path by another shortest path (see Fig.2.1 on Page 20 in Chapter 1, and Fig.3.1a on Page 42 in Chapter 2). The optimal Steiner node is approximated by a local search upon the grid. Any m genomes are recursively partitioned into a series of three genome groups. For a median problem, the approximated solution is very close to the optimal one. However, the computational cost is usually high.

(3) MGR Greedy split algorithm [3] is designed based on a recursive greedy split strategy. For genomes $g_1, g_2, \cdots, g_m$, it first connects the three closest genomes, say g_1, g_2 and g_3, by shortest paths. Suppose $g_1, g_2, \cdots, g_{m-1}$ have been connected by shortest paths. It then finds a split edge with the split site for g_m among all these paths. Finally, g_m is connected to the split site by a shortest path (see Fig.3.1c on Page 42 in Chapter 2). The optimal Steiner nodes are approximated by the split sites. The algorithm can deal with both unichromosomal and multichromosomal genomes, but may miss the optimal tree because no improvement is made after the tree is generated. It may also generate a tree far from the optimal tree. Its computational cost may also be expensive.

1.3 Proposed algorithms

The previous algorithms are not efficient enough in both accuracy and computational cost. The breakpoint analysis has an unclear biological meaning. Both the grid search and the MGR greedy split algorithm are computationally expensive. The later algorithm may generate a tree far from the optimal one. Therefore, more efficient algorithms are needed. We have developed the following approximation algorithms: (1) the nearest path search algorithm [27] (Chapter 2), (2) the branch-and-bound algorithm [28] (Chapter 3), (3) the neighbor-perturbing algorithm [28] (Chapter 3), (4) perturbing-improving algorithm [29] (Chapter 4), and (5) partitioning algorithm [30] (Chapter 5).

(1) Nearest path search algorithm [27] searches for the optimal Steiner nodes of a multiple genome rearrangement by signed reversal. For three genomes: g_1, g_2, g_3, it at first connects the nearest pair, say g_1 and g_2, by a shortest signed reversal path P_1. Then P_1 is improved to $P_2, \cdots, P_k$ such that each P_j is constructed from $N(P_{j-1})$ (i.e. the neighborhood of P_{j-1}) and is closer to g_3. A grid is constructed by joining g_3 to each node in P_k. Finally, an optimal Steiner node is approximated by a local search upon the grid (see Fig.2.4 on Page 27 in Chapter 1, Fig.3.1b on Page 42 in Chapter 2, and Fig.4.1b on Page 62 in Chapter 3). Moreover, the local search can be improved to search upon a band with fixed length and width (see Fig.2.4 on Page 27 in Chapter 1). Any m genomes are recursively processed by a series of groups consisitng three genomes.

This nearest path search algorithm improves Sankoff's grid search algorithm [25] by putting the local search near the optimal Steiner node (ancestral genome) upon a simpler grid. It increases the chance to find the optimal ancestral genome and decreases the computational cost (Sankoff's grid search algorithm has a running time at least $O(n^6)$, comparing to $O(n^3)$ for our nearest path search algorithm).

(2) Branch-and-bound algorithm for an median problem [28] is designed to search for the optimal Steiner nodes by checking all candidate nodes that satisfy certain

candidate conditions. S denotes the set of all possible candidates and is initialized. For each $s \in S$, check all $x \in N(s)$ (i.e. the neighborhood of s). If x satisfies the candidate conditions, x is chosen as a new candidate and is added into S. Repeat the process until convergence. The optimal Steiner node is the best one over S by minimizing the total reversal distance (see Fig.3.3 on Page 48 in Chapter 2). This algorithm is very efficient for small scale similar genomes.

(3) Neighbor-perturbing algorithm [28] searches for optimal Steiner nodes by perturbing Steiner nodes nearby their neighborhoods and improving them until convergence. It has two steps: initialization and iteration. For a median problem, the genomes $G = \{g_1, g_2, g_3\}$. In the initialization step, g_2 is chosen as the initial Steiner node, i.e. $s_0 = g_2$. In the iteration step, each s_i is perturbed into a better one $s_{i+1} \in N(s_i)$ (i.e. $d^*(s_{i+1}, G) < d^*(s_i, G)$, where $d^*(x, G) = \sum_{i=1}^{3} d(x, g_i)$, which is the total signed reversal distance). The Steiner nodes are improved from $s_0(= g_2)$, to $s_1, s_2, \cdots, s_i, \cdots$. Finally, s_i converges at s (see Fig.3.2a and Fig.3.2b on Page 46 in Chapter 2).

For m genomes $G = \{g_1, g_2, \cdots, g_m\}$, a weighted graph is defined by G and all signed reversal distances $d(g_i, g_j)$. In the initialization step, a minimum spanning tree T is chosen as the initial Steiner tree. The initial Steiner nodes, denoted by a set S, consists of all given genomes. In the iteration step, for each Steiner node $s \in S$, check all $x \in N(s)$. If $S \cup \{x\} - \{s\}$ generates a better minimum spanning tree than S does, replace s by x in S (see Fig.3.2c on Page 46 in Chapter 2). The Steiner nodes are improved from the initial ones until convergence.

This algorithm is much better than Bourque and Pevzner's MGR greedy split algorithm: Our neighbor-perturbing algorithm is a polynomial $2-$approximation algorithm (i.e. the total length of the tree obtained by the neighbor-perturbing algorithm is always within twice of the optimal one [28]) and has a lower computational cost $O(m^2n^4)$. However, Bourque and Pevzner's MGR greedy split algorithm may generate an approximate

solution far from the optimal one and has an expensive computational cost at least $O(m^3n^7)$. Therefore, our neighbor-perturbing algorithm is more efficient than MGR greedy split algorithm.

(4) Perturbing-improving algorithm for reconstructing ancestral genomes is designed to search for the optimal ancestral genomes. Its goal is to improve the neighbor-perturbing algorithm [29] so that it is more efficient than Bourque and Pevzner's MGR greedy split algorithm and has more chance to find optimal trees.

The neighbor-perturbing algorithm can be improved in three ways. For any existing Steiner node s, a better Steiner node x is sought in the neighborhood $N(s)$. Searching over a bigger neighborhood $N_p(s)$ for some $p \geq 1$ will increase the chance to find such a better x. Second, if more than one new x is chosen in each search, it should greatly increase the chance to find the optimal Steiner nodes. Third, any new x is chosen under the strict condition $d^*(x, G) < d^*(s, G)$, which may cause the algorithm to miss the optimal Steiner node. We suggest instead the condition is modified to $d^*(x, G) \leq d^*(s, G)$.

Based on these three factors, we design the perturbing-improving algorithm. The algorithm consists of three steps. Initially, the algorithm connects all given genomes by a tree with no internal node. Then, for each given genome, an internal node is created and assigned by an initial ancestral genome. Finally, the ancestral genomes are recursively improved until convergence or a fixed number of iteration steps. During recursive iteration, each ancestral genome may be updated to several new ones chosen from a bigger neighborhood at least as good as the old one.

This is a $2-$approximation algorithm. Because each ancestral genome may be respectively updated by several new genomes that are at least as good as the old one, the algorithm has more chances to find the optimal ancestral genomes and improves the neighbor-perturbing algorithm. The neighbor-perturbing algorithm can find only one

optimal solution. However, the perturbing-improving algorithm can find more than one optimal solution if exist. It is shown by both simulated and biological data that the algorithm is more efficient than MGR greedy split algorithm [29].

(5) Partitioning algorithm is designed to process large scale of genomes. Because all current existing algorithms can only efficiently reconstruct phylogenetic trees for small scale of genomes. For a large scale of genomes (which may contain a large number of genomes and each genome may contain a large number of genes), all algorithms are not efficient enough. If the number of genomes is large, the accuracy of the inferred tree may be low. On the other hand, if the number of genes is large, the computational cost will be high.

We design a partitioning algorithm to partition a large collection of genomes into a series of small groups whose evolutionary histories can be efficiently reconstructed by current existing algorithms. The evolutionary history of the large collection of genomes is approximated by joining all evolutionary histories of the small groups by minimizing the total distance [30].

Our algorithms are shown to be more efficient by theoretical proofs, computer simulations, and biological applications. The algorithms are implemented into programs (e.g. *GenomeNP, GenomeBnB, Mgenome*) and are available on the Internet at

"http://xgu.zool.iastate.edu".

1.4 Further work in the future

There is much interesting and challenging further work on multiple genome rearrangement. The most interesting part is to improve both the approximation algorithms and the mathematical model.

(1) Improve the algorithms

Our algorithms always search for better Steiner nodes over the whole neighborhood of old Steiner nodes. The computational cost is expensive. A random search strategy may reduce the computational cost. Meanwhile, the accuracy of inferred tree can be assured by reasonable random search strategies.

(2) Improve the model

The model of multiple genome rearrangement can be improved in two ways. At first, our discussion is based on pure signed reversals. However, evolution involves many other types of mutation. So we need to extend the model to handle more types of mutations (i.e. evolutionary events), such as insertion, deletion, substitution, reversal, translocation, fusion, fission, etc.

In addition, we assume genomes contain only one copy of each gene. In practice, genomes often contain multiple copies of genes. The further discussion should allow multiple gene copies.

Therefore, we need to discuss a more general multiple genome rearrangement problem: For a given collection of genomes, each of which may contain multichromosomes and multiple copies of its genes, reconstruct the evolutionary history under a collection of mutations (evolutionary events) including insertion, deletion, substitution, reversal (signed or unsigned), transposition, translocation, fusion, fission, recombination, gene duplication, gene loss, etc. The problem is NP-hard and efficient approximation algorithms are the best ways to solve the problem. Algorithm development becomes more challenging.

1.5 Dissertation organization

The rest of this dissertation consists of five chapters, each of which is a paper that describes algorithms for various cases of multiple genome rearrangement by signed reversal. In Chapter two, the nearest-path search algorithm is designed. In Chapter three, the neighbor-perturbing algorithm and branch-and-bound algorithm are developed. In Chapter four,we generalize the neighbor-perturbing and branch-and-bound algorithms into the perturbing-improving algorithm. In Chapter five, we present the partitioning algorithm for large scale genomes. Finally, Chapter six gives a brief summary of all the algorithms.

References

[1] Bader, D. A., Moret, B. M. E. and Yan, M., (2001). *A linear-time algorithm for computing inversion distances between signed permutations with an experimental study.* Lecture Notes in Computer Science, 2125: 365-376.

[2] Blanchette, M., Bourque, G., and Sankoff, D. (1997). *Breakpoint phylogenies.* In *Genome Information Workshop (GIW 1997),* (eds. Mivano S. and Takagi, T.), pp.25-34. University Academy Press, Tokyo.

[3] Bourque, G. and Pevzner, P. (2002). *Genome-Scale Evolution: Reconstructing Gene Orders in the Ancestral Species.* Genome Research 12: 26-36.

[4] Caprara, A. (1997). *Sorting by Reversals is Difficult.* Proceedings of the First Annual International Conference on Computational Molecular Biology (RECOMB'97), ACM Press, New York.

[5] Caprara, A. (1999). *Formulations and hardness of multiple sorting by reversals.* Proceedings of the Third Annual International Conference on Computational Molecular Biology (RECOMB'99), ACM Press, New York.

[6] Durbin, R., Eddy, S. R., Krogh, A. and Mitchison, G. (1998). *Biological sequence analysis: Probabilistic models of proteins and nucleic acids*, Cambridge University Press, Cambridge, UK.

[7] Felsenstein, J. (1981). *Evolutionary trees from DNA sequences: a maximum likelihood approach.* Journal of Molecular Evolution, 17:368-376.

[8] Fitch, W.M. (1971). *Toward defining the course of evolution: minimum change for a specific tree topology.* Systems Zoology, 35:406-416.

[9] Gu, X., (1999). *Statistical methods for testing functional divergence after gene duplication.* Molecular Biology and Evolution 16:1664-1674.

[10] Gu, X., (2000). *A simple evolutionary model for genome phylogeny inference based on gene content.* Comparative genomics (ed. Sankoff and Nadeau), pp.515-524. Kluwer Academic Publishers, The Netherlands .

[11] Gu, X., (2001). *Maximum likelihood approach for gene family evolution under functional divergence.* Molecular Biology and Evolution. 18: 453-464.

[12] Gusfield, D. (1997). *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, Cambridge, UK.

[13] Korkin, D. and Goldfarb, L., (2002). *Multiple genome rearrangement: A General approach via the evolutionary genome graph.* Bioinformatics, V18: pp.303-311.

[14] Lake, J. A., (1994). *Reconstructing evolutionary trees from DNA and protein sequences: paralinear distances.* Proceeding of National Science, USA, 91:1455-1459.

[15] Moret, B. M. E., Wyman, S., Bader, D. A., Warnow, T., and Yan, M. (2001). *A new implementation and detailed study of breakpoint analysis,* Proceeding of 6th Pacific Symposium on Biocomputing (PSB 2001), Hawaii, World Scientific Pub., 583-594.

[16] Ohno, S. (1967). *Sex Chromosomes and Sex-Linked Genes.* Berlitn SpringerVerlag.

[17] Otu, H. H. and Sayood, K. (2003). *A new sequence distance measure for phylogenetic tree construction.* Bioinformatics, 19:2122-2130.

[18] Sankoff, D. and Blanchette, M. (1998). *Multiple genome rearrangement and breakpoint phologeny.* Journal of Computational Biology, 5: 555-570.

[19] Sankoff, D., Leduc, G., Antoine, N., Paquin, B., Lang, B. F. and Cedergen,R. (1992). *Gene order comparisons for phylogenetic inference: Evolution of the mitochondrial genome.* Proceeding of National Science, USA, 89:6575-6579.

[20] Snel, B., Bork, P. and Huynen, M. A. (1999). *Genome phylogeny based on gene content.* Nature Genetics, 21: 108-110.

[21] Snel, B., Bork, P. and Huynen, M. A. (2000). *Genome evolution: gene fusion versus gene fission.* Trends Genetics, 16: 9-11.

[22] Snel, B., Bork, P. and Huynen, M. A. (2002). *Genomes in flux: the evolution of archaeal and proteobacterial gene content.* Genome Research, 12:17-25.

[23] Saitou, N. and Nei, M. (1987). *The neighbor-joining method: a new method for reconstructing phylogenetic trees.* Molecular Biology Evolution, 4: 406-425.

[24] Sankoff, D. (1999). *Genome rearrangement with gene families.* Bioinfomatics, 15:909-917.

[25] Sankoff, D., Sudaram, G. and Kececioglu, J. (1996). *Steiner points in the space of genome rearrangements.* International Journal of the Foundations of Computer Science, 7:1-9.

[26] Watterson, G. A., Ewens, W. J., Hall, T. E. and Morgan, A., (1982). *The chromosome inversion problem,* Journal of Theoretical Biology, 99:1-7.

[27] Wu, S. and Gu, X., (2002). *Multiple Genome Rearrangement By Reversals.* Pacific Symposium on Biocomputing 7:259-270.

[28] Wu, S. and Gu, X., (2003). *Algorithms for Multiple Genome Rearrangement by Signed Reversals.* Pacific Symposium on Biocomputing 8:363-374.

[29] Wu, S. and Gu, X. (2004). *An efficient algorithm for reconstructing ancestral genomes* , to be submitted.

[30] Wu, S. and Gu, X. (2004). *A partitioning algorithm for multiple genome rearrangement by signed reversals*, to be submitted.

CHAPTER 2. Multiple Genome Rearrangement by Reversals

A paper published in

Pacific Symposium on Biocomputing 7(2002), pp.259-270.

Shiquan Wu and Xun Gu

Abstract

In this paper, we discuss a multiple genome rearrangement problem: Given a collection of genomes represented by signed permutations, we generate the collection from some common ancestral genome (e.g. the identity permutation) in the minimum number of signed reversals. The problem is NP-hard, so efficient heuristics is an important way to find its optimal solution. At first, we discuss the problem for several special cases, including generating two genomes and three genomes by polynomial-time algorithms. Then we design a nearest path search algorithm to generate two genomes in general. Finally, we obtain an approximation algorithm to generate multiple genomes. We also show by experimental examples that the algorithms are efficient.

2.1 Introduction

Comparative genomics is one of the most important areas in bioinformatics and computational biology. Sorting by reversal plays a central role in comparative genomics. The problem was originated in last decade [2, 3, 4]. Its goal is to determine the evolutionary distances between organisms by inversions in genomes. Early, transformations of genomes are widely studied under evolutionary events such as insertion, deletion, point mutation (substitution), etc [11, 12]. Recently, the discussion is extended to other mutations, such as breakpoint, reversal, translocation[3, 4, 11, 12, 18, 19, 20], and recombination[21]. So far, most of the discussions on comparative genomics has been focused on sorting by reversal[2, 4, 5, 7, 8, 13, 14, 19], where a genome is represented by a signed permutation and an optimal reversal scenario (i.e. optimal reversal process) is found from any given permutation to the identity permutation.

Sorting by reversal is categorized into two classes: sorting by unsigned and signed reversals. At first, sorting by unsigned reversals is NP-hard[7, 8]. Therefore, only efficient approximation algorithms can be expected to find the optimal solution of the problem. So far, the best approximation algorithm has been a 1.5-approximation algorithm[10]. It is proved that there exists no polynomial-time 1.0008-approximation algorithm[6].

However, sorting by signed reversal is polynomial-time solvable[13, 14]. Several quadratic-time algorithms are widely used for finding the optimal solutions of the problem [5, 15, 16]. Recently, a linear-time algorithm is designed for computing the signed reversal distance between any two signed permutations[1].

Sorting by reversal can be regarded as a problem that generates a permutation from some given permutation by the minimum number of reversals. Multiple genome rearrangement by reversal is a generalization of sorting by reversal. It is to generate a given collection of permutations (i.e. genomes) from a common ancestral genome (e.g. the identity permutation) in the minimum number of reversals [17, 18]. For the un-

signed case, the problem is obviously NP-hard (since sorting by unsigned reversals is NP-hard[7, 8]). For the signed case, it is proved that the problem is NP-hard even if two permutations are generated from a common ancestral genome[8]. This implies that the problem is extremely hard. Therefore, it is interesting, also our purpose in this paper, to find efficient heuristics, or special cases that are polynomial-time solvable. Heuristics can be combinatorial or experimental algorithms [1, 9].

A similar genome rearrangement problem was discussed under reversal and transposition [17]. An approximation algorithm was designed to find the optimal solution by a local search upon a grid. In this paper, we discuss a multiple genome rearrangement problem for generating a collection of permutations from some fixed permutation (i.e. a common ancestral genome) in the minimum number of signed reversals. At first, we discuss the problem for several special cases, including generating two genomes and three genomes by polynomial-time algorithms. Then we design a nearest path search algorithm to generate two genomes in general. Finally, we obtain an approximation algorithm to generate multiple genomes. We also show by experimental examples that the algorithms are efficient.

The rest of the paper includes five parts: (1) Mathematical model, (2) Related problems, (3) Theorems and algorithms, (4) Experimental applications, (5) Discussion and future work.

2.2 Mathematical model

First of all, we introduce our main definitions and notations. The mathematical model of multiple genome rearrangement problem is described.

Definition 1 A genome is represented as a signed permutation $p = (p_1 p_2 \cdots p_{|X|})$ on the set X of genes. A signed reversal on the segment $[i, j]$ of p is defined as the

following operation that transforms a permutation p into another one $r(p;i,j)$:

$$\begin{aligned} p &= (p_1\ p_2\ \cdots\ p_{i-1}\ \underline{p_i\ \ p_{i+1}\ \ \cdots\ \ p_j}\ p_{j+1}\cdots\ p_{|X|}) \\ r(p;i,j) &= (p_1\ p_2\ \cdots\ p_{i-1}\ \underline{-p_j\cdots-p_{i+1}-p_i}\ p_{j+1}\cdots\ p_{|X|}) \end{aligned}$$

Definition 2 Let P be a collection of permutations. Define $N(P) = \{q|q = r(p;i,j)$ for some $p \in P, 1 \leq i \leq j \leq n\}$, called the reversal neighborhood of P. Define $N_1(P) = N(P)$, $N_2(P) = N(N_1(P))$, and $N_k(P) = N(N_{k-1}(P))$, the k-neighborhood of P.

Throughout the paper, genome and signed permutations are equivalent. An ancestral genome is represented by a Steiner node or internal node. A Steiner tree is equivalent to a phylogenetic tree. With the above definitions, our genome rearrangement problem is described as follows.

Multiple Genome Rearrangement By Signed Reversal (i.e. $(1,n)$ −MGRBSR) Given a collection of genomes $Q = \{q_1, q_2, \cdots, q_n\}$ on a set X of genes, which are represented by signed permutations, generate the collection from some common ancestral genome (e.g. the identity permutation $p = 12\cdots|X|$) in the minimum number of signed reversals, i.e. find a collection of signed permutations $t_r (1 \leq r \leq s)$ on X such that (1) any q_j is obtained from the common ancestral genome by a series of signed reversals $t_{r_1}, t_{r_2}, \cdots, t_{r_u}$, where each $t_{r_{j+1}}$ is obtained from t_{r_j} by one signed reversal, and (2) s is minimized.

We at first discuss $(1,2)-$MGRBSR problem. The general $(1,n)-$MGRBSR problem is then split into a series of $(1,2)-$MGRBSR problems.

2.3 Related problems

Our $(1, n)-$MGRBSR problem is similar or closely connected to the following problems:

(1) Multiple alignment[11, 12] Given some sequences, find the alignment with minimum pairwise score. In our $(1, n)-$MGRBSR problem, we do not consider the pairwise score, but the minimum total length of Steiner trees on the given permutations.

(2) Star alignment[11, 12] Given some sequences, find one median sequence such that the total alignment score between the median sequence and each given sequence is minimized. Our $(1, n)-$MGRBSR problem generalizes the problem and may contain a number of median/internal sequences.

(3) Fixed topology alignment[22] Given some sequences and a topological structure (usually, a tree) T. Each leaf of T is labeled by one given sequence. Assign one sequence to each internal node of T such that the total alignment score for all edges of T is minimized. Our $(1, n)-$MGRBSR problem is not restricted to a fixed topology and it is a topology-free alignment problem.

(4) Sorting by reversal[2, 4, 13, 14, 19] Given a permutation, transform it into the identity permutation in the minimum number of signed (or unsigned) reversals, i.e. it generates the identity permutation from a given permutation in the minimum number of signed (or unsigned) reversals. Our $(1, n)-$MGRBSR problem generalizes the problem to generating a collection of permutations from the identity permutation.

(5) Multiple genome rearrangement by reversal and transposition Sankoff

et al [17] discussed a multiple genome rearrangement problem under reversals and transpositions. For three genomes, the optimal solution was approximated by the local search upon a grid consisting of a series of paths from the three genomes (see Fig.2.1). A general problem with m genomes is recursively approximated by a series of groups of three genomes. We aim to design a grid search algorithm to improve Sankoff's local search on the grid (see Fig.2.4).

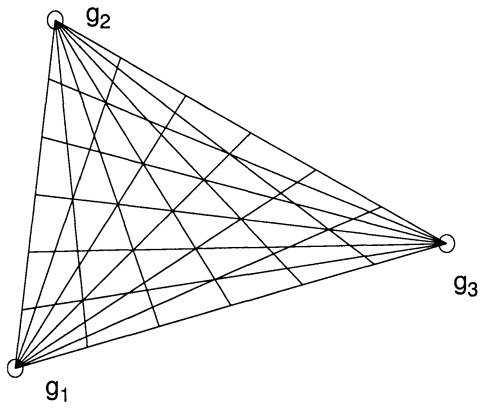

Figure 2.1 Sankoff's grid search algorithm.

2.4 Theorems and algorithms

In this part, we find algorithms for the $(1,n)-$MGRBSR problem. Our discussion goes from the special cases of $(1,2)-$, $(1,3)-$MGRBSR problem to the general problem. If a $(1,2)-$MGRBSR problem contains a pair of close permutations, we get a polynomial-time algorithm. If a $(1,3)-$MGRBSR problem consists of two pairs of close permutations, we get another polynomial-time algorithm. Based on these polynomial-time algorithms, we design our nearest path search algorithm for the general $(1,2)-$MGRBSR problem. And finally, we split a $(1,n)-$MGRBSR problem into

a series of $(1,2)-$MGRBSR problems and obtain an approximation algorithm for the general $(1,n)-$MGRBSR problem.

First of all, we have

Theorem 1 $(1,1)-$MGRBSR problem is solvable in a running time $O(|X|)$ [1].

A linear-time algorithm is presented for computing the reversal distance between two signed permutations [1] (however, finding the optimal reversal scenario still has a running time $O(|X|^2)$). We denote it BMY algorithm and will use it in our algorithms.

We can easily have the following approximation algorithm. At first, we construct a weighted graph with all given permutations as its vertices. All pairs of the given permutations form its edges. The weight of an edge is defined as the reversal distance of the pair of permutations representing the edge, which is computed by the BMY algorithm in a linear-time. Then, we find a minimum weight spanning tree of the graph. And finally, we generate all permutations from a given permutation (common ancestral genome) along the edges of the spanning tree.

Algorithm A

Input	Sequences: $p, q_1, q_2, \cdots, q_n$.
Output	Reversal process (i.e. scenario).
Step 1	Apply the BMY algorithm to construct a graph $G = (V, E, W)$ with $V = \{p, q_1, q_2, \cdots, q_n\}$, $E = \{[u,v] \mid u, v \in V, u \neq v\}$, and $W([u,v]) = d(u,v)$, which is the reversal distance between u and v.
Step 2	Find a minimum weight spanning tree T of G.
Step 3	Generate all permutations from p along the edges of T.

Theorem 2 Algorithm A finds an approximated solution for any $(1,n)-$MGRBSR problem in a running time $O(n^2|X| + n|X|^2)$.

Proof Step 1 has a running time $O(n^2|X|)$ since it takes $O(|X|)$ time to find each $W([u,v])$ and G has $O(n^2)$ edges. Step 2 has a running time $O(n\ log\ n)$ to find T. Step 3 has a running time $O(n|X|^2)$ since it takes a running time $O(|X|^2)$ to find the optimal reversal process (i.e. scenario) for each edge $[u,v]$ and there are $n-1$ edges in T. It then follows the total running time.

Algorithm A is a trivial approximation. The number of reversals can be decreased by introducing some median permutations. Suppose we want to generate q_1 and q_2 from p. We at first generate a median permutation q_0 from p, then generate q_1 and q_2 from q_0, respectively. When q_0 is properly chosen, the number of reversals can be improved. The median permutation q_0 is called a Steiner permutation. If we want to generate a collection of permutations, multiple Steiner permutations will be chosen so as to minimize the total reversal distance. These Steiner permutations are called optimal if they minimize the total reversal distance. For a $(1,n)-$MGRBSR problem, there may be $n-1$ optimal Steiner permutations.

In order to find an optimal Steiner permutation q_0 for a $(1,2)-$MGRBSR problem, we can check each permutation on X and finally get an optimal one. However, there are $|X|!$ permutations, so the running time is at least $O(|X|!)$. Here, we find some special cases that are polynomial-time solvable.

Theorem 3 (1) Let $V=\{p,q_1,q_2\}$. Assume q_0 is an optimal Steiner permutation. Then $d(x,q_0)\leq d(x,y)$ for any $x,y\in V$ $(x\neq y)$, i.e. for any $x\in V$, the optimal Steiner permutation is the nearest one from x (see Fig.2.2).

(2) If $V=\{p,q_1,q_2\}$ contains a nearest pair with a reversal distance at most k, then an optimal Steiner permutation q_0 can be found in a running time $O(|X|^{2k+1})$ (see Fig.2.3).

(3) If $V=\{p,q_1,q_2,q_3\}$ consists of two nearest pairs with reversal distances at most k, then the two optimal Steiner permutations can be found in a running time $O(|X|^{4k+1})$

(see Fig.2.3).

Proof (1) By contradiction. Suppose that $d(q_2, q_0) > d(q_2, q_1)$. Then since $d(p, q_0) + d(q_0, q_1) \geq d(p, q_1)$, it follows that $d(p, q_0)+d(q_0, q_1)+d(q_0, q_2) > d(p, q_1)+d(q_1, q_2)$, which is a contradiction because pq_1 and q_1q_2 form a tree better than the optimal Steiner tree formed by $q_0q_i(i = 1, 2, 3)$.

(2) By (1), the optimal Steiner permutation $q_0 \in N_k(x)$ for some $x \in V$. Since $|N(x)| \leq |X|^2$ and $|N_k(x)| \leq |X|^2|N_{k-1}(x)| \leq |X|^{2k}$. We need a running time $O(|X|)$ to find $d(y, x)(y \in N_k(x))$. Therefore, the total running time is $O(|X|^{2k+1})$.

(3) Suppose that $d(p, q_1) \leq k$ and $d(q_2, q_3) \leq k$. By (2), for each pair $q_{01} \in N_k(p)$ and $q_{02} \in N_k(q_3)$, compute the total reversal distance $d(q_{01}, p) + d(q_{01}, q_1) + d(q_{01}, q_{02}) + d(q_{02}, q_2) + d(q_{02}, q_3)$. We then take the best pair q_{01} and q_{02} as the optimal Steiner permutations. By (2), the total running time is $O(|X|^{4k+1})$.

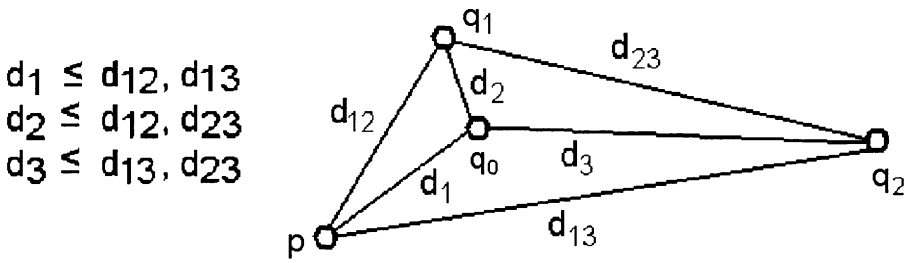

Figure 2.2 The optimal Steiner node q_0 is the nearest one from or to other nodes.

Based on Theorem 3, we can see from (1) that an optimal Steiner permutation is closer to the pair of permutations with the minimum distance than to the other permutations. Therefore, in (2), we find an optimal Steiner permutation in the k-neighborhood of a permutation in the nearest pair for a $(1, 2)-$MGRBSR problem. In (3), for a $(1, 3)-$ MGRBSR problem, we can also respectively find two optimal Steiner permutations in the k-neighborhoods of two permutations, each of which is in a nearest pair. We have the following algorithms (see Fig.2.3).

Algorithm B1 $(1,2)-$MGRBSR

Input Sequences: p, q_1, q_2

(a nearest pair with reversal distance at most k).

Output Optimal reversal process.

Step 1 Find the pair, say p and q_1, with the minimum reversal distance (at most k).

Step 2 Loop over all $u \in N_k(p)$ and update the reversal distance $d = d(u,p) + d(u,q_1) + d(u,q_2)$ and $q_0 = u$ if a better one is found.

Step 3 Generate q_0 from p, q_1 and q_2 from q_0 by BMY algorithm.

Algorithm B2 $(1,3)-$MGRBSR

Input Sequences: p, q_1, q_2, q_3

(two nearest pairs with reversal distances $\leq k$)

Output Optimal reversal process.

Step 1 Find the pairs, say p, q_1 and q_2, q_3, with two minimum reversal distances (at most k).

Step 2 Loop over $u \in N_k(p), v \in N_k(q_3)$. Update the total reversal distance $d = d(u,v) + d(u,p) + d(u,q_1) + d(v,q_2) + d(v,q_3)$ if a better one is found. Also update $q_{01} = u$ and $q_{02} = v$.

Step 3 Generate q_{01} from p, q_1 from q_{01}, q_{02} from q_{01}, q_1 and q_2 from q_{02} by the BMY algorithm.

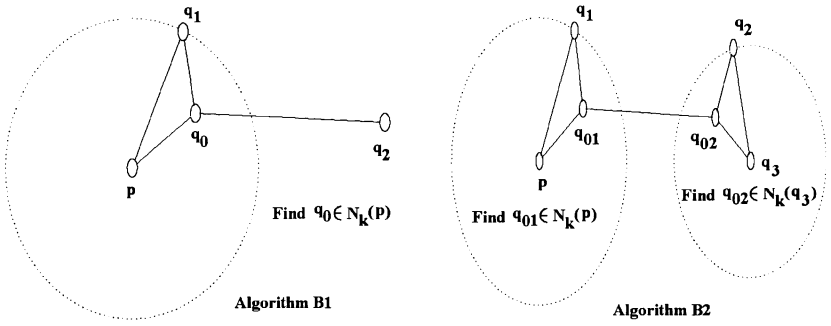

Figure 2.3 Algorithm B1/B2: optimal Steiner permutation is located in some $N_k(x)$

In Algorithm B1 and B2, each optimal Steiner permutation corresponds to one nearest pair of genomes. Because Theorem 3 shows that for any three genomes, the Steiner permutation is closer to the nearest pair of genomes. Based on this idea, we connect the two nearest genomes by a shortest reversal path and then begin to search for the optimal Steiner permutation from the neighborhood of the path. We have the following nearest path search algorithm.

Algorithm D Nearest path search

Input Sequences: p, q_1, q_2.

Output Steiner permutation and reversal process.

Step 1 Choose a minimum reversal distance pair, say p, q_1.

Step 2 Find the optimal reversal path P_1 from p to q_1.

Step 3 For $i \geq 1$, construct P_{i+1} from the neighborhood of P_i as follows such that P_{i+1} is closer to q_2:

(3.1) Find $M_i \in P_i$ minimizing $d(M_i, p, q_1, q_2) = d(M_i, p) + d(M_i, q_1) + d(M_i, q_2)$.

(3.2) Denote $P_i(M_i, w)$ a segment of P_i centered at M_i with length w. Find $M \in N_k(P_i(M_i, w))$ $(k \geq 1)$ minimizing $d(M, p, q_1, q_2) = d(M, p) + d(M, q_1) + d(M, q_2)$.

(3.3) P_{i+1} is the optimal reversal path from p to M, then to q_1.

Step 4 For each u in P_d (the last path in Step 3), find an optimal reversal path W from u to q_2. We get $W_1, W_2, \cdots, W_t$. Furthermore, from $W_1, W_2, \cdots, W_t$, construct a band with length l and width w centered at M_d. See Fig.2.4.

Step 5 For each x in all W_i (or the $l \times w$ band), do a local optimal search to find $q_0 \in N_k(x)$ minimizing $d(q_0, p, q_1, q_2)$.

Step 6 Choose the best q_0 in Step 5 as the optimal solution.

Step 7 Generate q_0 from p, q_1 and q_2 from q_0 by BMY algorithm.

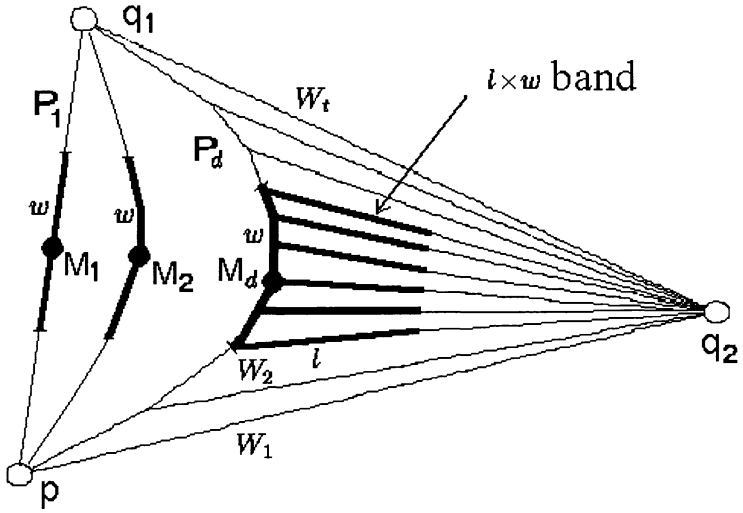

Figure 2.4 Algorithm D: (1) Find P_1 joining p and q_1. (2) Find M_1 in P_1 nearest to q_2. (3) Find M from the neighborhood of a segment of P_1 nearby M_1. (4) Find P_2 joining p, M, and then q_1. (5) From P_2, recursively find $P_3, \cdots, P_d$. (6) By q_2 and a segment of P_d nearby M_d, find W_i and construct a $l \times w$ band. (7) Do a local optimal search upon all W_i (or the $l \times w$ band).

In fact, in Algorithm D, we find a series of paths from p to q_1 such that they get closer and closer to q_2. Then construct a grid by using q_2 and the closest path to q_2. Finally, do local optimal searches on the grid or the $l \times w$ band. This nearest path search algorithm improves Sankoff's local search on grid [17] (also see Fig.2.1).

Theorem 4 For any p, q_1, q_2, the approximated Steiner permutation q_0 and reversal process can be found in a polynomial running time by Algorithm D.

Proof P_1 can be found in a running time $O(|X|^2)$ and it has at most $O(|X|)$ nodes. Since any reversal distance can be computed in a running time $O(|X|)$, $M_1 \in P_1$ can be found in a running time $O(|X|^2)$. For each node x of P_1, $N_k(x)$ has $O(|X|^{2k})$ nodes. It follows that $N_k(P_1(M_1, w))$ has at most $O(|X|^{2k})$ nodes. It then takes a running time $O(|X|^{2k+1})$ to find the node M. P_2 connects p, M, and q_1. Repeat the process to find P_3

and other paths. Finally, P_d can be found in d iterations (d is at most $|X|$). Therefore, the total running time is $O(|X|^{2k+2})$.

Based on Algorithm D, i.e. the nearest path search algorithm, we design an approximation algorithm to find the optimal Steiner permutations for the $(1,n)-$ MGRBSR problem. The main idea is to split the $(1,n)-$MGRBSR problem into a collection of $(1,2)-$MGRBSR problems, or to connect the permutations one after another into a tree. For any permutations: $\{p, q_1, q_2, \cdots, q_n\}$, we at first find a closest pair, say p and q_1 (i.e. they have the minimum signed reversal distance). Next, find a third permutation, say q_2, closest to $\{p, q_1\}$. Then apply Algorithm D to find a Steiner permutation s_1 for $\{p, q_1, q_2\}$ (see Fig.2.5(a)). Suppose $\{p, q_1, q_2, \cdots, q_i\}$ have been connected by a tree with the Steiner nodes $\{s_1, s_2, \cdots, s_{i-1}\}$. Then we find an un-connected permutation, say q_{i+1}, closest to $\{p, q_1, q_2, \cdots, q_i\}$. Re-label the permutations such that q_i is closest to q_{i+1}. Find an edge $q_i s_k$ of the tree such that s_k is closest to q_{i+1}. Re-label $\{s_1, s_2, \cdots, s_{i-1}\}$ such that $s_k = s_{i-1}$. Apply Algorithm D to find an optimal Steiner permutation s_i for $\{s_{i-1}, q_i, q_{i+1}\}$ (see Fig.2.5(b)). Repeat the process until all permutations $\{p, q_1, q_2, \cdots, q_n\}$ are connected by a tree.

Algorithm E **Recursive connecting**

Input Permutations $P = \{p, q_1, q_2, \cdots, q_n\}$.

Output Optimal Steiner permutations and reversal process.

Step 1 Initial tree: Connect the first three permutations.

(1.1) Find a closest pair, say p and q_1 (with the minimum signed reversal distance).

(1.2) Find a third permutation, say q_2, closest to $\{p, q_1\}$, i.e. q_2 and p (or q_1) have the minimum signed reversal distance.

(1.3) Apply Algorithm D to find an optimal Steiner permutation s_1 for $\{p, q_1, q_2\}$.

Step 2 Growing tree: Connect a new permutation to the tree.

(2.0) Suppose $\{p, q_1, q_2, \cdots, q_i\}$ have been connected by a tree with the Steiner nodes $\{s_1, s_2, \cdots, s_{i-1}\}$.

(2.1) Find a closest pair, say q_i and q_{i+1} (where q_i has been connected to the tree, but q_{i+1} has not been connected).

(2.2) Find an edge, say $q_i s_{i-1}$, such that s_{i-1} is closest to q_{i+1}.

(2.3) Apply Algorithm D to find an optimal Steiner permutation s_i for $\{s_{i-1}, q_i, q_{i+1}\}$.

Now, $\{p, q_1, q_2, \cdots, q_i, q_{i+1}\}$ have been connected by a tree with the Steiner nodes $\{s_1, s_2, \cdots, s_{i-1}, s_i\}$.

Step 3 Repeat Step 2 until all permutations $\{p, q_1, q_2, \cdots, q_n\}$ are connected by a tree.

Step 4 Find the reversal process along the tree.

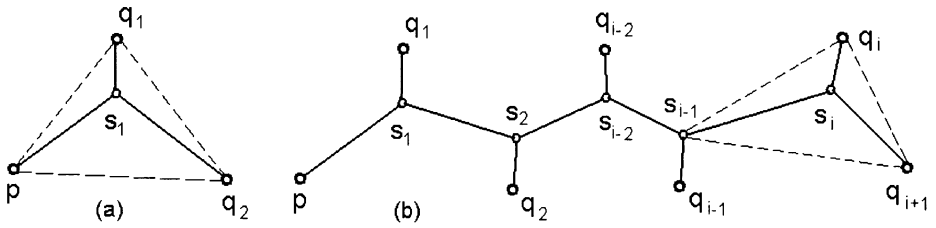

Figure 2.5 Algorithm E: (a) Step 1: Apply Algorithm D to find the optimal Steiner permutation s_1 for $\{p, q_1, q_2\}$. Where $d(p, q_1) \leq d(x, y)$ for all permutations x and y. $d(q_1, q_2) \leq d(q_1, y)$ for all permutations $y(\neq p)$. (b) Step 2: Apply Algorithm D to find the optimal s_i for for $\{s_{i-1}, q_i, q_{i+1}\}$. Where $d(q_i, q_{i+1}) \leq d(q_u, q_v)$ for all $u \leq i$ and $v \geq i+1$. $d(s_{i-1}, q_{i+1}) \leq d(s_u, q_{i+1})$ for all $u \leq i-1$.

Theorem 5 Algorithm E finds the approximated optimal Steiner permutations and the reversal process in a polynomial running time.

2.5 Experimental applications

Based on our algorithms, we design a computer program. Applying the program to some specific permutations, we obtain the optimal Steiner permutations for the permutations with different lengths. These examples show that our approximation algorithms are efficient, i.e. they find approximate solutions that are very close to the optimal ones. The three permutations, p, q_1, q_2, are chosen from the genomes of human, sea urchin, and fruit fly, respectively.

$p =$ 26 13 17 12 $-$ 24 15 18 $-$ 2 $-$ 16 $-$ 3 4 $-$ 28 7 5 1 10 19 25 22
11 29 14 20 $-$ 21 $-$ 8 6 30 $-$ 23 9 27.

$q_1 =$ 26 4 25 22 5 1 $-$ 28 19 11 29 20 $-$ 21 6 9 27 8 30 23 $-$ 24 16
14 $-$ 2 3 15 $-$ 7 10 13 17 12 18.

$q_2 =$ $-$26 $-$ 27 12 $-$ 24 15 18 $-$ 3 4 13 5 7 1 10 19 2 25 16 29 8 9
$-$20 $-$ 11 $-$ 22 30 23 21 6 28 17 $-$ 14.

By our program, we obtain an optimal Steiner permutation.

$q_0 =$ $26 - 2 - 14 - 29 - 11 - 22 - 25 - 19 - 10 - 1 - 5$ 13 17

12 $-$ 24 15 18 $-$ 7 28 $-$ 4 3 16 20 $-$ 21 $-$ 8 6 30 $-$ 23 9 27.

If we choose the first k genes of p, q_1, q_2 and apply the program for $k =$ 5, 10, 15, 20, 25, then we obtain the optimal Steiner permutations $q_0(k)$ from $p(k), q_1(k), q_2(k)$, respectively.

$p(5) =$ -2 -3 4 5 1.

$q_1(5) =$ 4 5 1 -2 3.

$q_2(5) =$ -3 4 5 1 2.

$q_0(5) =$ -2 -1 -5 -4 3.

$p(10) =$ -2 -3 4 7 5 1 10 -8 6 9.

$q_1(10) =$ 4 5 1 6 9 8 -2 3 -7 10.

$q_2(10) =$ -3 4 5 7 1 10 2 8 9 6.

$q_0(10) =$ -10 -1 -5 -7 -4 3 2 -8 6 9.

$p(15) =$ 13 12 15 -2 -3 4 7 5 1 10 11 14 -8 6 9.

$q_1(15) =$ 4 5 1 11 6 9 8 14 -2 3 15 -7 10 13 12.

$q_2(15) =$ 12 15 -3 4 13 5 7 1 10 2 8 9 -11 6 -14.

$q_0(15) =$ 13 12 15 $-$ 2 $-$ 10 $-$ 1 $-$ 5 $-$ 7 $-$ 4 3 11 $-$ 9 $-$ 6 8 $-$ 14.

$p(20) =$ 13 17 12 15 18 $-$ 2 $-$ 16 $-$ 3 4 7 5 1 10 19 11 14 20 $-$ 8 6 9.

$q_1(20) =$ 4 5 1 19 11 20 6 9 8 16 14 $-$ 2 3 15 $-$ 7 10 13 17 12 18.

$q_2(20) =$ 12 15 18 $-$ 3 4 13 5 7 1 10 19 2 16 8 9 $-$ 20 $-$ 11 6 17 $-$ 14.

$q_0(20) =$ $-20 - 14 - 11 - 19 - 10 - 1 - 5 - 7 - 4$ 3 16 8 13 17

12 15 18 $-$ 2 6 9.

$p(25) =$ 13 17 12 $-$24 15 18 $-$2 $-$16 $-$3 4 7 5 1 10 19 25 22 11
14 20 $-$21 $-$8 6 $-$23 9.

$q_1(25) =$ 4 25 22 5 1 19 11 20 $-$21 6 9 8 23 $-$24 16 14 $-$2 3 15
$-$7 10 13 17 12 18.

$q_2(25) =$ 12 $-$24 15 18 $-$3 4 13 5 7 1 10 19 2 25 16 8 9 $-$20 $-$11
$-$22 23 21 6 17 $-$14.

$q_0(25) =$ 2 $-$18 $-$15 24 $-$12 $-$17 $-$13 25 22 11 20 $-$21 5 1 10
19 $-$14 $-$16 $-$3 4 7 $-$8 6 $-$23 9.

The following table shows the reversal distances of the optimal Steiner trees in the cases of various permutation lengths we consider:

Permutation length n	$d(p, q_1)$	$d(p, q_2)$	$d(q_1, q_2)$	Optimal $d(p, \{q_1, q_2\})$
5	3	2	3	4
10	8	8	9	15
15	12	9	14	19
20	15	13	19	26
25	19	18	24	33
30	21	22	29	40

The optimal reversal distances are computed by our program. They are almost the same as the lengths of the Steiner trees in a metric space. For example, for $n = 10$, we have $(d_1, d_2, d_3) = (8, 8, 9)$. In the Euclidean metric space, we compute the optimal Steiner tree and find that its length is 14.5. The optimal $d(q_0, p)+d(q_0, q_1)+d(q_0, q_2) = 15$. Both are close each other.

Our approximation algorithms can find the optimal solutions for most collections of genomes. In most cases, they are more efficient than Sankoff's local grid search[17].

2.6 Discussion and future work

In this paper, we discuss a $(1,n)-$MGRBSR problem. We design polynomial-time algorithms for several special cases. The nearest path search algorithm is designed for the case of three genomes. An efficient approximation algorithm for the general $(1,n)-$MGRBSR problem is then designed based on the nearest path search algorithm. The $(1,n)-$MGRBSR problem is one of the most important problems in comparative genomics. We are interested in designing more efficient approximation algorithms for finding optimal solutions for the general $(1,n)-$MGRBSR problem. The problem is very similar to Steiner tree problems in a metric space. With the application of Steiner tree theory, the problem can be solved efficiently. On the other hand, stochastic can also be applied to the discussion of the $(1,n)-$MGRBSR problem. This will be the subject of our future work.

Acknowledgment This work is supported by the NIH grant RO1 GM62118 (to X.G.) and Wu is supported in part by NSF of China (19771025).

References

[1] Bader, D. A., Moret, B. M. E. and Yan, M. (2001). *A linear-time algorithm for computing inversion distances between signed permutations with an experimental study.* Proceeding of 7th Workshop on Algorithms and Data Structures (WADS 01).

[2] Bafna, V. and Pevzner, P. (1994). *Genome rearrangements and sorting by reversals.* In Proceeding of 34th IEEE Symp. of the Foundations of Computer Science, 148-157. IEEE Computer Society Press, California.

[3] Bafna, V. and Pevzner, P. (1995). *Sorting permutations by transpositions.* Proceedings of the 6th Annual Symposium on Discrete Algorithms, pages 614-623. ACM Press, New York.

[4] Bafna, V. and Pevzner, P. (1996). *Genome rearrangements and sorting by reversals.* SIAM Journal on Computing, 25(2):272-289.

[5] Berman, P. and Hannenhalli, S. (1996). *Fast sorting by reversal.* Proceeding of Combinatorial Pattern Matching (CPM), 168-175. Also Lecture Notes in Computer Science.1075.

[6] Berman, P., and Karpinski, M. (1998). *On some tighter inapproximability results.* Technical Report TR98-065, ECCC.

[7] Caprara, A. (1997). *Sorting by Reversals is Difficult.* Proceedings of the First Annual International Conference on Computational Molecular Biology (RECOMB'97), ACM Press, New York.

[8] Caprara, A. (1999). *Formulations and hardness of multiple sorting by reversals.* Proceedings of the Third Annual International Conference on Computational Molecular Biology (RECOMB'99), ACM Press, New York.

[9] Caprara, A. and Lancia, G. (2000). *Experimental and Statistical Analysis of Sorting by Reversals.* in D. Sankoff and J.H. Nadeau (eds.) Comparative Genomics: Empirical and Analytical Approaches to Gene Order Dynamics, Kluwer Academic Publishers, The Netherlands .

[10] Christie, D. A. (1998). *A 3/2-approximation algorithm for Sorting by reversals.* Proceeding of 9th Ann. ACM-SIAM Symposium on Discrete Algorithms, ACM-SIAM, 244-252.

[11] Durbin, R., Eddy, S. R., Krogh, A. and Mitchison, G. (1998). *Biological sequence analysis: Probabilistic models of proteins and nucleic acids.* Cambridge University Press, Cambridge, UK.

[12] Gusfield, D. (1997). *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, Cambridge, UK.

[13] Hannenhalli, S. and Pevzner, P. (1995). *Transforming men into mice (polynomial-time algorithm for genomic distance problems.* Proceeding of IEEE Symposium of the Foundations of Computer Science.

[14] Hannenhalli, S. and Pevzner, P. (1995). *Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals).* Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing, 178-189.

[15] Kaplan, H., Shamir, R. and Tarjan, R. E. (1997). *Faster and simpler algorithm for sorting signed permutations by reversals.* Proceeding of eighth annual ACM-SIAM Symp. on Discrete Algorithms (SODA 97). ACM Press, New York.

[16] Kaplan, H., Shamir, R. and Tarjan, R. E. (2000). *Faster and simpler algorithm for sorting signed permutations by reversals.* SIAM Journal on Computing, 29(3):880-892.

[17] Sankoff, D., Sudaram, G. and Kececioglu, J. (1996). *Steiner points in the space of genome rearrangements.* International Journal of the Foundations of Computer Science, 7:1-9.

[18] Sankoff, D. and Blanchette, M. (1998). *Multiple genome rearrangement and breakpoint phologeny.* Journal of Computational Biology, 5: 555-570.

[19] Sankoff, D. and Nadeau, J. H. (2000). *Comparative Genomics: Empirical and Analytical Approaches to Gene Order Dynamics.* Kluwer Academic Publishers, The Netherlands .

[20] Watterson, G. A. and Ewens, W. J. and Hall, T. E. and Morgan, A. (1982). *The chromosome inversion problem.* Journal of Theoretical Biology, 99, 1-7.

[21] Wu, S. and Gu. X. (2001). *A greedy algorithm for optimal recombination.* Lecture Notes on Computer Science, 2108:86-90.

[22] Wang, L., Ma, B. and Li, M. (2000). *Fixed topology alignment with recombination.* Discrete Applied Mathematics. 104: 281-300.

CHAPTER 3. Algorithms for Multiple Genome Rearrangement by Signed Reversals

A paper published in

Pacific Symposium on Biocomputing 8(2003), pp.363-374.

Shiquan Wu and Xun Gu

Abstract

We discuss a multiple genome rearrangement problem by signed reversals: Given a collection of genomes, we generate them in the minimum number of signed reversals. It is NP-hard and equivalent to finding an optimal Steiner tree to connect the genomes by reversal paths. We design two algorithms to find the optimal Steiner nodes of the problem: Neighbor-perturbing algorithm and branch-and-bound algorithm. The first one is a polynomial running time approximation algorithm. It searches for the optimal Steiner nodes by perturbing initial Steiner nodes nearby their neighborhoods and improving them until convergence. The second one is an exact exponential running time algorithm for a median problem. It finds the optimal Steiner node by checking all candidates that satisfy the necessary conditions for optimal Steiner nodes. We implement the algorithms into two programs respectively and show by experimental examples that they are more efficient than other similar ones, such as GRAPPA, BPAnalysis, and MGR, etc.

3.1 Introduction

Phylogenetic trees show the evolutionary relationship of species. They can be inferred based on DNA or amino acid sequences. Three major methodologies are widely used for the purpose: Parsimony method, maximum likelihood method, and distance-based method [10, 11]. Genome projects have generated enormous types of informative genome data for phylogeny reconstructions [9]. One interesting and challenging problem is about comparative genome-wide gene orders for multiple genomes and phylogeny inference. Most earlier discussions focus on edit distances defined by local mutations such as insertion, deletion, substitution [10, 11], and furthermore recombination [18]. Recently, distance was widely discussed based on the orders of genes. Breakpoint analysis first investigated such kind of distances [17, 14, 15]. However, the biological meaning of breakpoint distance is not very clear. In contrast, reversal distance overcomes the problem. It is defined as the minimum number of signed/unsigned reversals needed to account the difference of the gene orders of two genomes [2, 3, 14, 15]. Sorting by reversal is an important problem in comparative genomics. It transforms one genome into another by signed or unsigned reversals [2, 3, 4]. Sorting by unsigned reversals is NP-hard [7, 8], while sorting by signed reversals is polynomial-time solvable [12, 13]. Indeed, the signed reversal distance between any two signed permutations can be computed by linear running time algorithms [1].

A multiple genome rearrangement problem discusses how to reconstruct optimal distance-based phylogenetic trees for a collection of genomes [15, 16, 19]. The problem is NP-hard [8]. Therefore, the practical purpose turns out to find good approximate solutions. Various approximation algorithms/programs (such as GRAPPA, BPAnalysis, MGR, etc.) are developed for solving different versions of multiple genome rearrangement problems [5, 6, 15, 16, 19].

In this paper, we discuss a multiple genome rearrangement problem by signed re-

versals [19]: Given a collection of genomes, we generate them in the minimum number of signed reversals. It is NP-hard and equivalent to finding an optimal Steiner tree to connect the genomes by reversal paths. We focus on designing efficient algorithms to find the optimal Steiner nodes for the genomes.

The rest of the paper consists of five parts. In Section 2, we introduce the mathematical model of our multiple genome rearrangement problem. In Section 3, we review some related previous algorithms on the problem. In Section 4, we design an approximation algorithm and an exact algorithm for the problem. In Section 5, we compare our algorithms/programs with other similar ones (such as GRAPPA, BPAnalysis, and MGR, etc.) and show that ours are more efficient than others. And finally, we discuss some possible further work and applications in Section 6. We obtain the following results.

(1) Design a neighbor-perturbing algorithm. It is a polynomial running time approximation algorithm and searches for the optimal Steiner nodes for all given genomes by perturbing initial Steiner nodes nearby their neighborhoods and improving them until convergence. It is a 2-approximation algorithm.

(2) Find the necessary conditions for optimal Steiner nodes.

(3) Based on the necessary conditions, we design a branch-and-bound algorithm. It is an exact exponential running time algorithm for a median problem and finds the optimal Steiner node by checking all possible candidates that satisfy the necessary conditions for optimal Steiner nodes.

(4) Two programs are implemented from the algorithms, respectively. Experimental examples show that our algorithms/programs are more efficient than other similar ones, such as GRAPPA, BPAnalysis, and MGR, etc.

3.2 Problem and model

First of all, we introduce the notations and set up the mathematical model of the multiple genome rearrangement problem by signed reversals [19].

Definition (1) Let X denote an alphabet (e.g. the set of genes) and $|X| = n$. Assume $p = (p_1 p_2 \cdots p_{i-1} \underline{p_i p_{i+1} \cdots p_j} p_{j+1} \cdots p_n)$ is a signed permutation (i.e. genome) on X. Each p_i stands for a gene and its sign represents its strand of DNA. A positive (or negative) sign means that the gene is in the forward (or backward) strand. A signed reversal on a segment $[i, j]$ of p is defined as the operation

$$r(p; i, j) = (p_1 p_2 \cdots p_{i-1} \underline{-p_j \cdots - p_{i+1} - p_i} p_{j+1} \cdots p_n),$$

where the segment $[i, j]$ changes both its orientation and signs (i.e. strand).

(2) Define $N(p) = \{q | q = r(p; i, j) \text{ for all } 1 \leq i \leq j \leq n\}$, which is the collection of all permutations that can be obtained from p by one signed reversal. $N(p)$ is called a reversal neighborhood (or sphere) of p. Let P be a collection of permutations. A reversal neighborhood (or sphere) of P is defined as $N(P) = \cup_{p \in P} N(p)$. Define $N_1(P) = N(P)$, $N_2(P) = N(N_1(P))$, and $N_k(P) = N(N_{k-1}(P))$, the $k-$neighborhood (or $k-$sphere) of P.

MGRBSR Problem (Multiple Genome Rearrangement By Signed Reversal) Given a collection of permutations $G = \{g_1, g_2, \cdots, g_m\}$, we generate G from some $p \in G$ in the minimum number of signed reversals, i.e. to find a collection of signed permutations $t_k (1 \leq k \leq s)$ on X such that (1) any g_j is obtained from p by a series $t_{k_1}, t_{k_2}, \cdots, t_{k_j}$, where each $t_{k_{i+1}}$ is obtained from t_{k_i} by one signed reversal, i.e. $t_{k_{i+1}} \in N(t_{k_i})$, and (2) s is minimized. Denote $d(p, G) = s$.

Various kinds of multiple genome rearrangement problems are widely discussed. The

problems are NP-hard [8] and equivalent to finding optimal Steiner trees for G in the space of permutations [6, 10, 14, 15, 16, 19]. We here discuss the **MGRBSR Problem** involving only pure signed reversals.

3.3 Previous related work

Many algorithms are designed to solve various kinds of multiple genome rearrangement problems. Most of them are approximation algorithms.

Breakpoint analysis The breakpoint distance between two genomes is defined as the number of consecutive pairs of genes that are adjacent in one genome but not in the other. Breakpoint analysis is one of the earliest methods for genome rearrangement problem based on gene orders. There are several efficient approximation algorithms and programs for the problems, such as GRAPPA and BPAnalysis [5, 15, 17]. However, breakpoint analysis gives many approximation solutions that do not have a clear biological meaning. It could not find a more biological accurate rearrangement (reversal) distance [6].

Grid search Sankoff *et al* [16] discussed a multiple genome rearrangement problem for reversals and transpositions. For three genomes, a local optimal solution was searched upon a grid consisting of a series of reversal paths (see Fig.2.1 on Page 20). A general problem with m genomes is recursively approximated by a series of groups of three genomes.

Nearest path search Wu and Gu [19] discussed a multiple genome rearrangement problem for pure signed reversals. An approximate solution was obtained by a nearest path search algorithm upon a simple grid (see Fig.3.1a). For three genomes, suppose g_1 and g_2 are the closest pair. The algorithm at first finds a shortest reversal path P_1 from g_1 to g_2, then improves P_1 to another better reversal path P_2 (where P_2 is constructed from the nodes in the neighborhood of P_1 and closer to g_3 than P_1), and repeatedly improves

a series of reversal paths to get a closest reversal path P_k (from the neighborhood of P_{k-1}) to g_3. Next, it constructs a grid by P_k and g_3. Finally, a local optimal solution is obtained by searching on the grid. This simplifies Sankoff's *grid search* algorithm [16] and is shown to be efficient by experimental examples.

Greedy split Bourque and Pevzner [6] designed a multiple genome rearrangement algorithm based on recursively greedy splitting. For the given genomes $g_1, g_2, \cdots, g_m$, the algorithm first connects the three closest genomes, say g_1, g_2 and g_3, by shortest paths. Suppose $g_1, g_2, \cdots, g_{m-1}$ have been connected by some reversal paths. Then the algorithm finds a split edge with a split site from all these paths and connects g_m to the split site (see Fig.3.1b). The algorithm can be applied to both unichromosomal and multichromosomal genomes.

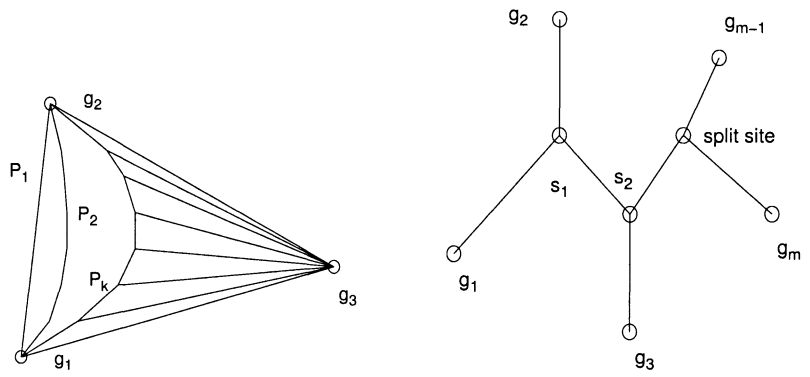

Figure 3.1 (a) Nearest path search (b) Greedy split

3.4 Approximation algorithms

In this section, we design two algorithms: Neighbor-perturbing algorithm and branch-and-bound algorithm. Multiple genome rearrangement by signed reversals can be efficiently solved by applying these two algorithms.

3.4.1 Neighbor-perturbing algorithm

Any minimum spanning tree is a 2-approximation solution of the optimal Steiner tree (to prove this, add a multiple edge for each edge of the optimal Steiner tree T^* and compare a spanning tree T_0 with these edges. Then we have an inequality on the lengths of the trees: $L(T_{MinSpanningTree}) \leq L(T_0) \leq 2L(T^*)$). Any minimum spanning tree can be regarded as a trivial Steiner tree with each given g_i as a trivial Steiner node. Within $N(g_i)$, we can find some Steiner node better than g_i. For a median problem, we at first choose g_2 as an initial Steiner node s_0. Then find a Steiner node $s_1 \in N(s_0)$ better than s_0, and next find $s_2 \in N(s_1)$ better than s_1, and so on. The Steiner nodes are improved better and better. This is the idea of our neighbor-perturbing algorithm. Neighbor-perturbing algorithm is to perturb an initial Steiner node within its neighborhood and improve it better and better until convergence.

The neighbor-perturbing algorithm consists of two steps: Initialization and iteration. For a median problem, in the initialization step, we rearrange the genomes so that the path $g_1g_2g_3$ forms a minimum spanning tree. g_2 is then chosen as the initial Steiner node, i.e. $s_0 = g_2$. We start perturbing from g_2 so that the iteration converges fast. In the iteration step, we perturb each s_i and find a better $s_{i+1} \in N(s_i)$ (Fig.3.2b). The Steiner nodes are improved better and better from $s_0(= g_2)$ to $s_1, s_2, \cdots, s_i, \cdots$, until s_i finally converges at s (Fig.3.2a).

Algorithm **Neighbor-perturbing (Median problem)**

Input Permutations: $G = \{g_1, g_2, g_3\}$.

Output Optimal Steiner node.

Step 1 Initialization:

(1.1) Rearrange each g_i: $d(g_1, g_3) = max\{d(x, y)|x, y \in G\}$.

(1.2) Initial Steiner node $s_0 = g_2$.

Step 2 Iteration:

(2.1) Suppose s_i have been defined. Check each $x \in N(s_i)$:

Denote $d^*(x, G) = d(x, g_1) + d(x, g_2) + d(x, g_3)$.

If x is better than s_i, i.e. $d^*(x, G) < d^*(s_i, G)$,

then define $s_{i+1} = x$.

(2.2) Repeat (2.1) until convergence.

For the general multiple genome rearrangement problem, in the initialization step, we choose a minimum spanning tree T as an initial Steiner tree. We rearrange the genomes so that g_1 and g_m are the two longest leaves in T. There are at most $m - 2$ Steiner nodes. If we perturb g_1 and g_m to get two Steiner nodes, it may take a longer time for the convergence. So we choose $g_2, g_3, \cdots, g_{m-1}$ as the initial Steiner nodes so as to get a better approximate solution and a fast convergence. In the iteration step, we perturb and improve each Steiner node s by checking all $x \in N(s)$. If $S \cup \{x\} - \{s\}$ generates a better Steiner tree than S does, then replace s by x (Fig.3.2c). The Steiner nodes/tree are improved from the initial ones until convergence.

Algorithm **Neighbor-perturbing (General case,** see Fig.3.2c)

Input Permutations: $G = \{g_1, g_2, \cdots, g_m\}$.

Output Optimal Steiner nodes S.

Step 1 Initialization:

(1.1) Define the reversal distance graph for G (vertex set) by all pairwise signed reversal distances $d(x, y)(x, y \in G)$.

(1.2) Find a minimum spanning tree and its two longest leaves x_0 and y_0. Choose the initial $S = G - \{x_0, y_0\}$.

Step 2 Iteration:

(2.1) For each $s \in S$, check all $x \in N(s)$:

If the optimal Steiner tree generated by $S \cup \{x\} - \{s\}$ is better than that generated by S, then replace s by x.

(2.2) Repeat (2.1) until convergence.

Theorem 1 *The Neighbor-perturbing algorithm is a 2-approximation algorithm and has a polynomial running time.*

Proof The algorithm is obviously 2-approximation.

For the running time, Step (1.1) has a running time $O(m^2n)$ since there are $O(m^2)$ edges and each edge needs a running time $O(n)$ for finding its reversal distance. Step (1.2) has a running time $O(mlogm)$ for finding the minimum spanning tree. It follows that Step 1 has a running time $O(m^2n)$. In Step (2.1), m reversal distances are updated and each reversal distance needs a running time $O(n)$. $N(s)$ has $O(n^2)$ nodes. The total running time of Step (2.1) is $O(mn^3)$. Finally, for each iteration of Step (2.1), the total length of the tree will decrease for at least one. The tree has $m-1$ edges and the length of each edge is at most $O(n)$. Therefore the total decreasing is at most $O(mn)$. It follows that Step 2 terminates within a running time $O(m^2n^4)$. Therefore, the algorithm has a running time $O(m^2n^4)$.

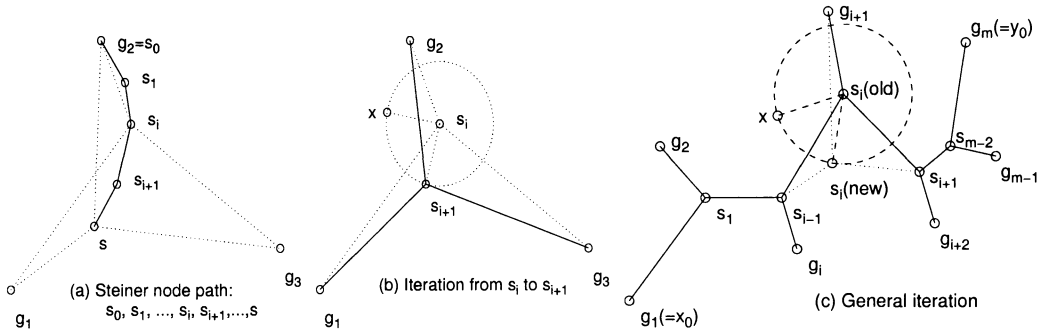

Figure 3.2 (a) Initial Steiner node $s_0 = g_2$. Recursively iterate/update the Steiner nodes: $s_0, s_1, \ldots, s_i, \cdots$ until convergence (at s). (b) Process of iteration from s_i to s_{i+1}: Suppose s_i have been generated. Check each $x \in N(s_i)$, if x is better than s_i, then update $s_{i+1} = x$. If no better x is found, then return s_i (optimal). (c) General iteration: Assume g_1 and g_m are the two longest leaves in the minimum spanning tree. Then $g_2, g_3, \cdots, g_{m-1}$ are chosen as the initial Steiner nodes. Each s_i is generated from $g_{i+1}(i = 2, 3, \cdots, m-1)$ by a series of iterations. In each iteration, for Steiner node s_i, check all $x \in N(s_i)$, if $S \cup \{x\} - \{s_i\}$ generates a better Steiner tree, then replace s_i by x, i.e. s_i is updated by a new s_i. The three edges $s_{i-1}s_i$, $g_{i+1}s_i$, $s_{i+1}s_i$ are replaced by three new $s_{i-1}s_i$, $g_{i+1}s_i$, $s_{i+1}s_i$, respectively. The old Steiner tree is updated by a new one. The iterations are repeated until convergence.

3.4.2 Branch-and-bound algorithm

In this part, we design an exact algorithm, branch-and-bound algorithm, to find the optimal Steiner node for a median problem. Assume $G = \{g_1, g_2, g_3\}$. We rearrange the genomes so that the path $g_1g_2g_3$ forms a minimum spanning tree. Denote $d^*(x, G) = d(x, g_1) + d(x, g_2) + d(x, g_3)$. For any optimal Steiner node s, the corresponding optimal Steiner tree must at first connect s to some $s_1 \in N(g_2)$, and then connect s_1 to g_2 (see Fig.3.3a). $g_2s_1g_1$ and $g_2s_1g_3$ are two optimal reversal paths from g_2 to g_1 and g_3, respectively. They share a common part g_2s_1 ($s_1 \in N(g_2)$). We can see that $d(g_2, s_1) + d(s_1, g_i) \geq d(g_2, g_i)$, $d(s_1, g_i) \leq min\{d(g_1, g_2), d(g_1, g_3), d(g_2, g_3)\}(i = 1, 2, 3)$ and $d^*(s_1, G) \leq d^*(g_2, G)$, i.e. s_1 is a Steiner node at least as good as g_2.

Generally, any optimal Steiner tree connects s and g_2 through a series of Steiner

nodes $s_0(=g_2), s_1, s_2, \cdots, s_j, \cdots$ such that $s_j \in N(s_{j-1})(j \geq 0)$ (see Fig.3.3b). $s_0 s_1 s_2 \cdots s_j \cdots g_1$ and $s_0 s_1 s_2 \cdots s_j \cdots g_3$ are two optimal reversal paths from g_2 to g_1 and g_3, respectively. They share a common part $s_0 s_1 s_2 \cdots s_j$. Each s_q is a Steiner node at least as better as s_{q-1} for all $q \geq 1$.

Theorem 2 (Necessary conditions for an optimal Steiner node) *Let* $G = \{g_1, g_2, g_3\}$ *(genomes). If* s *is an optimal Steiner node, then*

$$\frac{1}{2}[d(g_1,g_2) + d(g_1,g_3) + d(g_2,g_3)] \leq d(s,g_i) \leq min\{d(g_1,g_2), d(g_1,g_3), d(g_2,g_3)\} \tag{3.1}$$

Moreover, there exist a series of Steiner nodes $s_0 = g_2, s_1, s_2, \cdots, s_j, \cdots, s_p = s$ *such that Eq. (3.1) holds for each* s_j *and* $d^*(s,G) \leq d^*(s_j,G) \leq d^*(s_{j-1},G) \leq d^*(s_0,G)$, $(2 \leq j < p)$.

Proof The first inequality holds because $d(g_i,s) + d(s,g_j) \geq d(g_i,g_j)$. The second inequality follows from the principle of the proof of Theorem 3 in [19], i.e. the optimal Steiner node is the nearest one to each g_i.

Theorem 2 gives us the idea of our branch-and-bound algorithm. Any s satisfying Eq. (3.1) is a candidate of the optimal Steiner node. By $s_j \in N(s_{j-1})(j \geq 0)$ and Eq. (3.1), we can recursively search for the optimal Steiner node from all possible candidates starting from g_2. At first, we check all $x \in N(g_2)$. If $s_1 \in N(g_2)$ satisfies Eq. (3.1), then it is one of the first generation candidates of g_2. Next, we find all second generation candidates s_2 from each first generation candidate s_1 by $s_2 \in N(s_1)$ and Eq. (3.1). We repeatedly find all candidates for all generations until convergence. The optimal Steiner node can be found by $min_j d^*(s_j,G)$ over all candidates s_j. The relations between all candidates of different generations can be expressed as a tree (see Fig.3.3c-d). Suppose at some iteration step we have found some candidates of some generations (see Fig.3.3c). Let B the set of all candidates (denoted by branches) and L the set of all the current

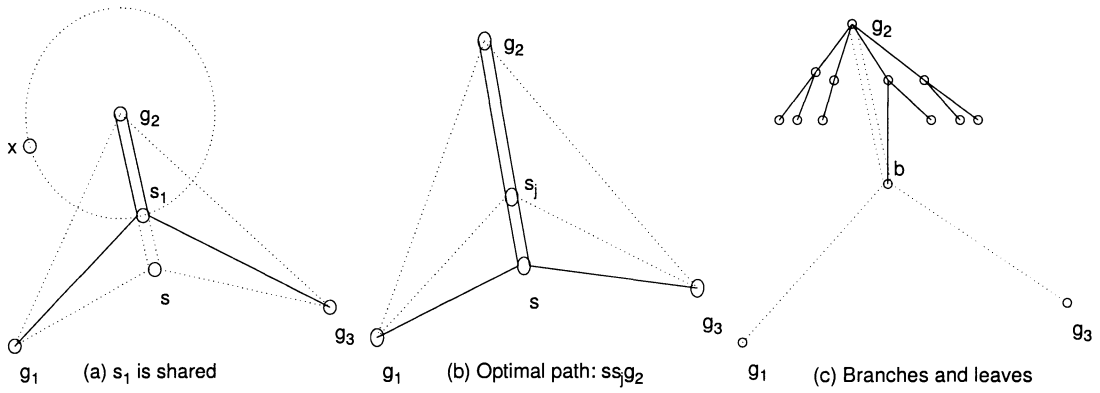

Figure 3.3 (a) Two optimal reversal paths $g_2 s_1 g_1$ and $g_2 s_1 g_3$ share a common part $g_2 s_1$. Eq. (3.1) holds for each s_1 and $d^*(s_1, G) \leq d^*(g_2, G)$. (b) s is an optimal Steiner node. $g_2 \cdots s_j \cdots g_1$ and $g_2 \cdots s_j \cdots g_3$ are two optimal reversal paths from g_2 to g_1 and to g_3, respectively. Eq. (3.1) holds for each s_j and $d^*(s, G) \leq d^*(s_j, G) \leq d^*(s_{j-1}, G) \leq d^*(s_0, G)$ for each j. (c)Generation relations are shown by the branches/leaves, each leaf is obtained from one branch (candidate) of last generation. (d)Process of growing branches: For each leaf b, check all $x \in N(b)$, if x satisfies Eq. (3.1), then take x as a new branch and a new leaf. Update s by x if $d^*(x, G) < d^*(s, G)$. New leaves $b_1, b_2, \cdots, b_k$ are found from b. b becomes an internal node in next iteration.

generations (denoted by leaves). For each leaf $b \in L$, check all $x \in N(b) - B$, if x satisfies Eq. (3.1), then x is a new candidate and it is then put into the branch set B and the leaf set L. If x is better than s, i.e. $d^*(x, G) < d^*(s, G)$, then update the optimal Steiner node s by x. New leaves (new generation candidates) $b_1, b_2, \cdots, b_k$ are obtained from b (see Fig.3.3d). b is removed from L and becomes an internal node in next iteration. We repeat the process until convergence, i.e. no new branch can be found.

Branch-and-bound algorithm consists of two steps: Initialization and iteration. In the initialization step, we start from g_2. In the iteration step, we grow branches (i.e. the candidate set) and update the optimal Steiner node (see Fig.3.3d).

Algorithm **Branch-and-bound** (Median) (see Fig.3.3)

Input Permutations: $G = \{g_1, g_2, g_3\}$.

Output Optimal Steiner node s.

Step 1 Initialization:

(1.1) Rearrange each g_i: $d(g_1, g_3) = max\{d(x, y)|x, y \in G\}$.

(1.2) Initial $B = \{g_2\}$, $L = \{g_2\}$, $s = g_2$.

Step 2 Iteration: (see Fig.3.3d)

(2.1) Grow branches and update the optimal Steiner node:

For each $b \in L$, check every $x \in N(b) - B$:

If Eq. (3.1) holds for x, then

x is a new candidate: $B = B \cup \{x\}$, $L = L \cup \{x\}$.

If $d^*(x, G) < d^*(s, G)$, then update $s = x$.

Remove b from L after all $x \in N(b) - B$ is checked.

(2.2) Repeat (2.1) until convergence.

Theorem 3 *The branch-and-bound algorithm finds the optimal Steiner node in an exponential running time.*

3.5 Comparisons and examples

We show that our algorithms are more efficient than other similar ones. Theoretically, our neighbor-perturbing algorithm is more efficient than MGR greedy-split algorithm [6] and other similar algorithms. The MGR greedy-split algorithm greedily connects all given genomes one after another. Usually, a greedy algorithm may generate an approximate solution far from the optimal one in the worse case. However, our neighbor-perturbing algorithm starts from a minimum spanning tree, which is a 2-approximation solution. The neighbor-perturbing algorithm improves each Steiner node

again and again and has more chances to get the optimal one. This is its advantage over other algorithms.

The two algorithms are implemented into two computer programs, GenomeNP and GenomeBnB, respectively, which take a reversal distance program "signed.c" as a subroutine written by Hannenhalli (http:// www-hto.usc.edu/software). We show by experimental examples that our algorithms/programs are more efficient than other similar ones, such as GRAPPA, BPAnalysis, MGR, etc.

Example 1 $G = \{p, q_1, q_2\}$ (the genomes of human, sea urchin,fruit fly) [6, 16].

$p =$ 26 13 17 12 -24 15 18 32 -2 -16 -3 -33 4 -28 7 5 1 10 19 25 22 11

29 14 20 -21 -8 6 30 -23 9 27 31

$q_1 =$ 26 4 25 22 5 1 -28 19 11 29 20 -21 6 9 27 8 30 23 -24 16 14 -2 32

3 -31 15 -7 33 10 13 17 12 18

$q_2 =$ 26 14 17 -28 -6 -21 23 -30 22 11 20 9 -8 -29 -16 -25 -2 -19 -10 -1

-7 -5 -13 -4 33 3 -32 -18 -15 24 -12 27 31

By GenomeNP, we obtain an optimal Steiner node:

$s =$ 26 13 17 12 -24 15 18 32 -2 -3 -33 4 14 20 -21 -8 6 -7 28 -29 -11

-22 -25 -19 -10 -1 -5 -16 30 -23 9 27 31

Therefore, $d(s,p) = 4, d(s,q_1) = 20, d(s,q_2) = 15$, and $d(p,\{q_1,q_2\}) \leq 39$. GRAPPA and BPAnalysis obtained $d(p,\{q_1,q_2\}) = 43$, **MGR** also obtained $d(p, \{q_1, q_2\}) \leq 39$ [5, 6, 15]. However, can 39 be improved better? **MGR** did not answer. Since $d(s,p) + d(s,q_1) \geq d(p,q_1) = 24$, $d(s,p) + d(s,q_2) \geq d(p,q_2) = 19$, $d(s,q_1) + d(s,q_2) \geq d(q_1,q_2) = 32$, so $d(p,\{q_1,q_2\}) \geq [(24+19+32)/2] = 38$. GenomeBnB checks all possible candidates and obtains $d(p,\{q_1,q_2\}) = 39$.

Example 2 $G = \{A, B, \cdots, K\}$ are the following the genomes of: human, asterina pectinifera, paracentrotus lividus, drosophila yakuba, artemia franciscana, albinaria coerulea, cepaea nemoralis, katharina tunicata, lumbricus terrestris, ascaris suum, onchocerca volvulus, respectively [6].

$A =$ 1 -32 17 2 23 12 3 20 6 30 7 8 21 31 24 9 -10 -18 11 33 -28
19 14 34 13 25 4 22 -29 26 5 35 -15 -27 -16 -36

$B =$ 1 30 7 2 23 12 3 -32 6 8 21 31 9 -10 11 19 14 18 33 -13 -5
-22 -4 -25 -20 -36 17 -26 34 -16 -35 15 -24 -27 29 -28

$C =$ 1 30 7 2 23 12 3 -32 6 8 21 31 9 -10 11 19 14 18 33 28 -29
27 24 -15 35 16 -34 26 -17 36 20 25 4 22 5 13

$D =$ 1 25 2 23 17 12 3 20 6 15 30 27 31 18 -19 -9 -21 -8 -7 33
-28 10 11 32 -4 -24 -13 -34 -14 22 -29 26 5 35 -16 -36

$E =$ 1 25 2 23 17 12 3 20 6 15 30 27 31 18 -19 -9 -21 -8 -7 33
-28 10 11 32 -4 -24 -13 -34 -14 26 5 35 -22 -29 -16 -36

$F =$ 1 34 13 24 28 15 10 9 4 7 11 17 16 19 2 36 35 20 21 -29 -25
-27 -12 -30 -18 -14 -26 -6 -32 31 8 -33 -3 22 5 23

$G =$ 1 34 13 24 15 10 28 9 4 7 11 17 16 19 2 36 35 20 21 -29 -25
-27 -12 -30 -18 -14 26 -6 -32 -33 -3 31 8 22 5 23

$H =$ 1 17 2 12 -19 -9 -21 -8 -7 33 -32 -11 -10 28 -4 -25 -24 -13
-34 -14 -26 -16 -36 -35 -29 -20 -18 3 23 15 30 27 22 6 31 5

$I =$ 1 27 2 17 36 20 3 29 10 11 35 12 30 21 9 19 18 28 33 7 8 16
26 14 34 13 24 15 32 25 4 22 23 6 31 5

$J =$ 1 16 26 17 20 2 21 13 6 9 15 28 34 10 7 35 18 14 32 27 36 4
12 23 25 31 5 22 30 29 19 11 24 3 33 8

$K =$ 1 35 10 30 29 11 24 3 23 15 25 27 26 7 14 36 4 19 12 22 20 2
21 13 6 16 32 28 17 34 9 18 31 5 33 8

We at first construct the signed reversal distance graph for the genomes from all pairwise signed reversal distances (Fig.3.4a), then find a minimum spanning tree T of the graph (Fig.3.4b), and finally generate the following optimal Steiner nodes a, d, e, f, h, i

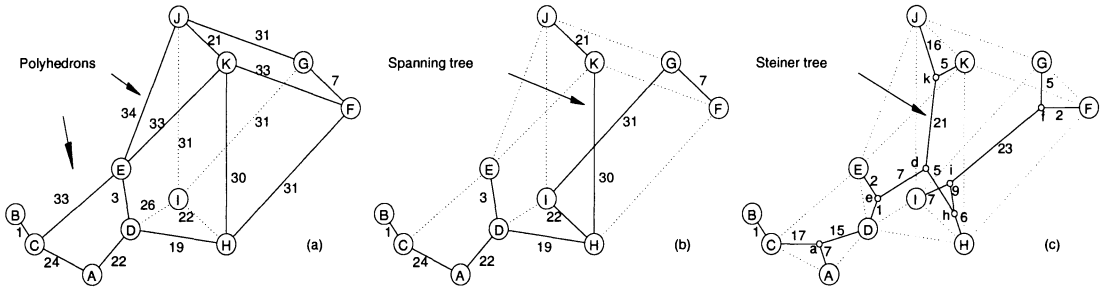

Figure 3.4 (a) The distance graph for the 11 genomes. (b) The minimum spanning tree of the genomes. (c) Optimal Steiner tree. The total distance 149 < 150 (obtained by **MGR**).

and k by perturbing T by GenomeNP (Fig.3.4c).

$a =$ 1 -32 17 -11 10 -9 -27 18 33 -28 19 14 34 13 25 4 22 -29 26 5 35

30 7 8 21 31 24 2 23 12 3 20 6 -15 -16 -36

$d =$ 1 25 31 5 -35 -29 -22 -12 -17 -23 -3 14 34 13 24 15 30 27 2 20 6

4 -32 -11 -10 28 -33 7 8 21 9 19 -18 -26 -16 -36

$e =$ 1 25 2 23 17 12 3 20 6 15 30 27 31 18 -19 -9 -21 -8 -7 33 -28 10

11 32 -4 -24 -13 -34 -14 22 29 26 5 35 -16 -36

$f =$ 1 34 13 24 15 10 9 4 7 11 17 16 19 2 36 35 20 21 -29 -25 -27 -12

-30 -18 -14 -26 -6 -32 31 8 -33 -28 -3 22 5 23

$h =$ 1 17 12 -19 -9 -21 -8 -7 33 -28 10 11 32 -4 -25 -23 -3 18 22 29

35 36 16 26 14 34 13 24 15 30 27 2 20 6 31 5

$i =$ 1 27 2 16 26 14 34 13 24 15 10 11 35 12 -18 -19 -9 -21 -8 -7 -33 -28

30 17 36 20 -4 -25 -23 -3 22 29 32 6 31 5

$k =$ 1 35 18 27 32 -9 -34 -17 -28 10 30 29 11 24 3 23 -7 -26 -16 -6 -13

-21 -2 -20 -22 -12 -19 -4 -36 -14 15 25 31 5 33 8

GenomeNP perturbs A, E, F, I, K, D, H into a, e, f, i, k, d, h, respectively. The minimum spanning tree (see Fig.3.4b, length: 180) is perturbed into the optimal Steiner

tree (Fig.3.4c) with a length 149, which is better than 150 obtained by **MGR** [6]. Our algorithms/programs are better than **MGR** and others.

3.6 Discussion

We have designed two algorithms (Neighbor-perturbing algorithm and branch-and-bound algorithm) and two programs (GenomeNP and GenomeBnB) to find the optimal Steiner nodes. The experimental examples show that the algorithms/programs are more efficient than other similar ones. Our discussions are based on signed reversals. However, we can extend the discussions to other mutations, such as insertion, deletion, substitution, reversal, transposition, translocation, fusion, fission, etc. The algorithms have great potential applications in a wide range of genomes including multichromosome genomes.

In the iteration steps of the algorithms, we always check all nodes in $N(x)$. We can get better approximate solutions by checking $N_k(x)$ for some $k \geq 2$. By randomly checking only some nodes in $N(x)$, or $N_k(x)$, we can also expect to get some satisfactory approximate solutions. Similarly, if we randomly check only some of the candidates, the branch-and-bound algorithm then becomes a nice approximation algorithm.

Acknowledgment This work is supported by the NIH grant RO1 GM62118 to X.G. and Wu is also is supported in part by NSF of China (19771025).

References

[1] Bader, D. A., Moret, B. M. E. and Yan, M., (2001). *A linear-time algorithm for computing inversion distances between signed permutations with an experimental study.* Lecture Notes in Computer Science, 2125: 365-376.

[2] Bafna, V. and Pevzner, P. (1994). *Genome rearrangements and sorting by reversals.* In Proceeding of 34th IEEE Symp. of the Foundations of Computer Science, 148-

157. IEEE Computer Society Press, California.

[3] Bafna, V. and Pevzner, P. (1995). *Sorting permutations by transpositions.* Proceedings of the 6th Annual Symposium on Discrete Algorithms, pages 614-623. ACM Press, New York.

[4] Bafna, V. and Pevzner, P. (1996). *Genome rearrangements and sorting by reversals.* SIAM Journal on Computing, 25(2):272-289.

[5] Blanchette, M., Bourque, G., and Sankoff, D. (1997). *Breakpoint phylogenies.* In *Genome Information Workshop (GIW 1997),* (eds. Mivano S. and Takagi, T.), pp.25-34. University Academy Press, Tokyo.

[6] Bourque, G. and Pevzner, P. (2002). *Genome-Scale Evolution: Reconstructing Gene Orders in the Ancestral Species.* Genome Research 12:26-36.

[7] Caprara, A. (1997). *Sorting by Reversals is Difficult.* Proceedings of the First Annual International Conference on Computational Molecular Biology (RECOMB'97), ACM Press, New York.

[8] Caprara, A. (1999). *Formulations and hardness of multiple sorting by reversals.* Proceedings of the Third Annual International Conference on Computational Molecular Biology (RECOMB'99), ACM Press, New York.

[9] Gu, X., (2000). *A simple evolutionary model for genome phylogeny inference based on gene content.* Comparative genomics (ed. Sankoff and Nadeau), pp.515-524. Kluwer Academic Publishers, The Netherlands .

[10] Gusfield, D. (1997). *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, Cambridge, UK.

[11] Hillis, D., Motitz, C., Mable, B. (1996). *Molecular systematics.* Sinauer Association, Inc. Massachusetts, USA.

[12] Hannenhalli, S. and Pevzner, P. (1995a). *Transforming men into mice (polynomial algorithm for genomic distance problems.* Proceeding of IEEE Symposium of the Foundations of Computer Science.

[13] Hannenhalli, S. and Pevzner, P. (1995b). *Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals).* Proceeding of of 27th Annual ACM Symposium on Theory of Computing, 178-189.

[14] Sankoff, D. (1999). *Genome rearrangement with gene families.* Bioinfomatics, 15:909-917.

[15] Sankoff, D. and Blanchette, M. (1998). *Multiple genome rearrangement and breakpoint phologeny.* Journal of Computational Biology, 5: 555-570.

[16] Sankoff, D., Sudaram, G. and Kececioglu, J. (1996). *Steiner points in the space of genome rearrangements.* International Journal of the Foundations of Computer Science, 7:1-9.

[17] Watterson, G. A. and Ewens, W. J. and Hall, T. E. and Morgan, A. (1982). *The chromosome inversion problem.* Journal of Theoretical Biology, 99, 1-7.

[18] Wu, S. and Gu, X., (2001). *A greedy algorithm for optimal recombination.* Lecture Notes on Computer Science, 2108:86-90.

[19] Wu, S. and Gu, X., (2002). *Multiple Genome Rearrangement By Reversals.* Pacific Symposium on Biocomputing 7:259-270.

CHAPTER 4. Reconstructing ancestral genomes

(A paper to be submitted)

Shiquan Wu and Xun Gu

Abstract

For a set of genomes (represented by signed permutations on genes), find a tree where the given genomes are assigned to leaf nodes and ancestral genomes are hypothesized at internal nodes such that the total reversal distance summed over all edges of the tree is minimized. The key is to reconstruct all optimal ancestral genomes for the internal nodes. The MGR algorithm proposed by Bourque and Pevzner finds a tree by "greedily" connecting the given genomes (see "Bourque, G. and Pevzner, P., Genome-Scale Evolution: Reconstructing Gene Orders in the Ancestral Species. *Genome Research* 12(2002): 26-36"). However, it is not always efficient and may generate a tree far from the optimal one. In this paper, a perturbing-improving algorithm is designed to search for the optimal ancestral genomes. Initially, the algorithm connects all given genomes by a tree with no internal node. Next, for each given genome, an internal node is created and assigned by an initial ancestral genome. Then, the ancestral genomes are recursively improved until convergence or a fixed number of iteration steps. This algorithm is shown to be more efficient than the MGR algorithm.

4.1 Introduction

The traditional phylogenetic tree in molecular evolution is inferred based on the analysis of individual genes (see Graur and Li 1997; Nei and Kumar 2000). Three major methods are widely used in practical applications: (1) **The sequence alignment method** finds all pairwise distances of sequences by sequence comparisons. The phylogenetic tree is then reconstructed from the matrix of the pairwise distances by various algorithms, e.g. neighbor-joining algorithm, UPGMA (Unweighted Pair Group Method using Arithmetic averages), and other algorithms (Sneath and Sokal 1973; Saitou and Nei 1987; Lake 1994; Gusfield 1997; Durbin *et al.* 1999). (2) **The parsimony method** finds a tree or topology that best fits or explains the biological data (Fitch 1971; Gusfield 1997; Durbin *et al.* 1999). (3) **The maximum likelihood method** uses probability models on nucleotide substitutions and Bayesian estimation to predict the best tree (Felsenstein 1981; Gu 1999,2000,2001; Gusfield 1997; Durbin *et al.* 1999). However, these three methods may generate incorrect phylogenies when organisms are very similar. The computational cost may be expensive for multiple sequence alignments.

With the rapid growth of genome data, the whole-genome approach becomes popular to extract the bulk phylogenetic signals for reconstructing the evolutionary history of the entire genome. In order to improve the accuracy of inference, people tried various new ways. For instance, Gu and Zhang (2004) formulated a stochastic framework for genome phylogeny inference by the (extended) gene content data including the absence, single-copy, or duplications of gene families across genomes. Other new methods are obtained by comparing genome content in various ways.

(4) **The Kolmogorov or Lempel-Ziv complexity method** is proposed to infer phylogenies by comparing gene contents or particular features, tandem repeats, and certain patterns occurring or not occurring in biological sequences (Snel *et al.* 1999, 2000, 2002; Otu and Sayood 2003). It increases the inference accuracy and decreases

the computational cost.

However, all these four methods deal with only local mutations. They may give rise to incorrect phylogenetic trees, since the evolutionary process involves both local and non-local mutations, such as insertion, deletion, substitution, reversal, translocation, etc. Therefore, the orders of genes should be taken into account for inferring phylogenies to estimate the extent of rearrangement events, comparative mapping, and the putative genomic architecture of ancestral genomes (Palmer and Herbon 1988; Palmer 1992; Sankoff et al. 1992; Olmstead et al. 1994; Bafna and Pevzner 1995; Hannenhalli et al. 1995; Blanchette et al. 1999; Cosner et al. 2000; Bourque and Pevzner 2002; Wu and Gu 2002, 2003). Various types of evolutionary edit distances are defined for transforming the gene order of one genome into another under both local and non-local mutations (Watterson *et al.* 1982; Sankoff *et al.* 1992, 1998, 1999). Breakpoint distance is the first of such distances (Watterson *et al.* 1982; Sankoff *et al.* 1998,1999), but it has an unclear biological meaning (Sankoff,1998; Bourque and Pevzner 2002). Reversal distance overcomes this problem. The reversal distance is defined as the minimum number of signed/unsigned reversals needed to account for the different gene orders of two genomes (Bafna and Pevzner, 1994 and 1995; Sankoff,1998; Sankoff et al., 1999). To find the reversal distance between two genomes/permutations is extensively discussed as sorting by reversal (Bourque et al. 1994, 1995, 1996; Bafna and Pevzner 1994, 1995; Hannenhalli and Pevzner 1995a, 1995b; Berman *et al.* 2001). It is an important problem in comparative genomics. Sorting by unsigned reversals is NP-hard (Caprara 1997, 1998). However, sorting by signed reversals is polynomial-time solvable (Hannenhalli and Pevzner, 1995a, 1995b; Berman et al. 1996; Sankoff 1999; Bader et al. 2001). Moreover, the signed reversal distance between any two signed permutations can be computed by linear running time algorithms (Berman *et al.* 1996; Sankoff 1999; Bader *et al.* 2001).

To generalize sorting by reversal for multiple genomes, we have a multiple genome rearrangement problem, which discusses how to reconstruct optimal distance-based phylogenetic trees for genomes (Hannenhalli and Pevzner 1995a, 1995b; Sankoff *et al.* 1996,1998; Bourque and Pevzner 2002; Wu and Gu 2002). The problem is NP-hard (Caprara 1999). Many approximation algorithms and programs, such as GRAPPA and BPAnalysis, are designed for solving different kinds of multiple genome rearrangement problems (Sankoff *et al.* 1996; Blanchette *et al.* 1997; Sankoff *et al.* 1998). MGR algorithm is proposed by Bourque and Pevzner (2002) to find a tree by "greedily" connecting given genomes. Although it has a better performance than previous algorithms/programs (e.g. GRAPPA and BPAnalysis), we have recognized that the tree reconstructed by "greedy split" may be far from optimal. Based on our previous work on multiple genome rearrangement by pure signed reversal (Wu and Gu 2002, 2003), in this paper, we develop a more efficient 2-approximation algorithm to search for the optimal ancestral genomes. Through theoretical comparison, simulation and biological data, we demonstrate that our new algorithm is more efficient than the MGR algorithm.

The rest of this paper consists of five parts. Section 4.2 describes the mathematical notation for multiple genome rearrangement by signed reversals. Section 4.3 reviews several previous algorithms on which our perturbing-improving algorithm is based. Section 4.4 designs the perturbing-improving algorithm. Section 4.5 shows that the algorithm is more efficient than MGR. Finally, Section 4.6 discusses some possible further work on the problem.

4.2 Mathematical model

A genome is viewed as a signed permutation $p = (p_1 p_2 \cdots p_n)$ on genes, where each p_i stands for a gene and its sign represents the coding strand of DNA, e.g. a positive (or negative) sign indicates the forward (or backward) strand. A signed re-

versal on a segment $p_i p_{i+1} \cdots p_j$ of p is defined as the operation that transforms $p = (p_1 p_2 \cdots p_{i-1}\ \underline{p_i p_{i+1} \cdots p_j} p_{j+1} \cdots p_n)$ into $r(p;i,j) = (p_1 p_2 \cdots p_{i-1}\ \underline{-p_j \cdots - p_{i+1} - p_i} p_{j+1} \cdots p_n)$, where each gene in the segment changes both its orientation and sign.

For any genomes p and q, the reversal distance $d(p,q)$ is defined as the minimum number of reversals that transform p into q. The corresponding reversals that transform p into q form a shortest reversal path between p and q.

For a genome p, the reversal neighborhood (or sphere) of p is defined as $N(p) = \{q | d(p,q) = 1\}$, which is the set of all signed permutations that can be transformed from p by a single signed reversal. For a set P of genomes, its reversal neighborhood (or sphere) is defined as $N(P) = \cup_{p \in P} N(p)$. Define $N_1(P) = N(P)$. For $k \geq 2$, $N_k(P) = N(N_{k-1}(P))$ is called the $k-$neighborhood (or $k-$sphere) of P.

Multiple Genome Rearrangement By Signed Reversal Given a collection of m genomes $G = \{g_1, g_2, \cdots, g_m\}$ (signed permutations), reconstruct their phylogenetic tree under signed reversal, i.e. find a tree where the given genomes are assigned to leaf nodes and ancestral genomes (i.e. signed permutations) are hypothesized at internal nodes such that the total reversal distance summed over all edges of the tree is minimized.

4.3 Previous algorithms

Multiple genome arrangement has been extensively discussed. Under reversal and transposition, the **grid search algorithm** (Sankoff *et al.* 1996) searches for local optimal ancestral genomes (i.e. Steiner nodes) upon a grid consisting of a series of paths, which connect one genome to another, or join the intermediate nodes of the paths to other genomes (see Fig.2.1 on Page 20, Wu and Gu 2002). The algorithm is efficient in most cases. However, the computational cost may be expensive. In order to reduce the computational cost, a **nearest path search algorithm** is designed to search for

the optimal ancestral genomes upon a simpler grid. For three genomes: g_1, g_2, g_3, the algorithm at first connects the nearest pair, say g_1 and g_2, by a shortest signed reversal path P_1. Then improves P_1 to $P_2, \cdots, P_k$ such that P_j is created from $N(P_{j-1})$ and is closer to g_3. A grid is constructed by joining g_3 to each node in P_k. Finally, an optimal Steiner node (i.e. ancestral genome) is approximated by a local search upon the grid (see Fig.3.1a on Page 42, Wu and Gu 2002). Any m genomes are recursively processed by a series of three genome groups. The algorithm is shown to be more efficient by experimental examples (Wu and Gu 2002).

MGR greedy split algorithm reconstructs a tree by "greedily" connecting all given genomes (Bourque and Pevzner 2002). Suppose $g_1, g_2, \cdots, g_m$ are the given genomes. The three closest genomes, say g_1, g_2 and g_3, are first connected by shortest reversal paths. If $g_1, g_2, \cdots, g_{i-1}$ have been connected by a partial tree, the algorithm joins one more genome, say g_i, to the split site of the split edge in the tree (see Fig.3.1b on Page 42, Wu and Gu 2003). The algorithm is not always efficient. The tree reconstructed by "greedy split" may be far from the optimal one.

Neighbor-perturbing algorithm (Wu and Gu 2003) searches for optimal Steiner nodes by perturbing Steiner nodes nearby their neighborhoods and improving them until convergence. It has two steps: Initialization and iteration. For a median problem, the genomes $G = \{g_1, g_2, g_3\}$. In the initialization step, g_2 is chosen as the initial Steiner node, i.e. $s_0 = g_2$. In the iteration step, each s_i is perturbed into a better one $s_{i+1} \in N(s_i)$ (i.e. $d^*(s_{i+1}, G) < d^*(s_i, G)$, where $d^*(x, G) = \sum_{i=1}^{3} d(x, g_i)$. See Fig.3.2b on Page 46, Wu and Gu 2003). The Steiner nodes are improved from $s_0 (= g_2)$, to $s_1, s_2, \cdots, s_i, \cdots$. Finally, s_i converges at s (see Fig.3.2a on Page 46, Wu and Gu 2003).

For m genomes $G = \{g_1, g_2, \cdots, g_m\}$, a weighted graph is defined by G and all pairwise reversal distances $d(g_i, g_j)$. In the initialization step, a minimum spanning tree

T is chosen as the initial Steiner tree. The initial Steiner nodes, denoted by a set S, consists of all given genomes (except for the two leaves, say g_1 and g_m, with the longest signed reversal lengths in T). In the iteration step, for each Steiner node $s \in S$, check all $x \in N(s)$. Replace s by x in S whenever $S \cup \{x\} - \{s\}$ generates a better minimum spanning tree than S (see Fig.3.2c on Page 46, Wu and Gu 2003). Improve the Steiner nodes from the initial ones until convergence.

4.4 Perturbing-improving algorithm

The previous algorithms are not efficient enough. Both the grid search algorithm (Sankoff *et al.*, 1996) and the nearest path search algorithm (Wu and Gu, 2002) are computationally expensive because of their local searching on grids. The greedy split algorithm (Bourque and Pevzner, 2002) may easily miss the optimal tree because it simply connects genomes by a "greedy-split" strategy and does not improve the tree after it is generated. The neighbor-perturbing algorithm (Wu and Gu, 2003) has been shown to be both more efficient and more accurate. However, it can still be improved in three ways. First, for any existing Steiner node s, a better Steiner node x is sought in the neighborhood $N(s)$. Searching over a bigger neighborhood $N_p(s)$ for some $p \geq 1$ will increase the chance to find such a better x. Second, if more than one new x is chosen in each search, it should greatly increase the chance to find the optimal Steiner nodes. Third, any new x is chosen under the strict condition $d^*(x, G) < d^*(s, G)$, which may cause the algorithm to miss the optimal Steiner node. We modify the condition to $d^*(x, G) \leq d^*(s, G)$.

Consider these three factors, we design a perturbing-improving algorithm that improves the neighbor-perturbing algorithm. Each ancestral genome may be updated by multiple new ancestral genomes that are chosen from a bigger neighborhood and at least as good as the old one.

Steps of algorithm

The algorithm consists of three steps:

(1) Initially, the algorithm connects all given genomes one after another by shortest reversal paths, which form a tree on the given genomes with no internal node. The connecting process is as follows.

First of all, find two genomes, e.g. g_1 and g_2, with the minimum reversal distance. Then connect the two genomes by a shortest reversal path. Suppose $g_1, g_2, \cdots, g_i$ have been connected by shortest reversal paths. Among all unconnected genomes, find the one, e.g. g_{i+1}, with the minimum reversal distance to all connected genomes $g_1, g_2, \cdots, g_i$. Suppose $d(g_{i+1}, g_j)$ is the minimum reversal distance ($1 \leq j \leq i$). Then connect g_{i+1} and g_j by a shortest reversal path. After all given genomes are connected, a minimum spanning tree is obtained for the given genomes.

(2) Next, for each given genome, an internal node is created and assigned by an initial ancestral genome.

Denote a_i the ancestral genome (i.e. the signed permutation) for the internal node created for g_i. Assign g_i to a_i as an initial ancestral genome. Then the initial ancestral genomes consists of all given genomes, denoted by a set S (to be improved).

(3) Finally, the ancestral genomes are recursively improved until convergence.

For each $a_i \in S$, check all signed permutations $x \in N_p(a_i)$, i.e. $d(x, a_i) \leq p$ ($p \geq 1$), and update each a_i by one of the following three strategies:

(3.1) **One-to-one improving** If the minimum spanning tree of $S \cup \{x\} - \{a_i\}$ is better (i.e. shorter) than that of S, update $a_i = x$. Recursively improve all a_i until convergence.

(3.2) **One-to-one random improving** If the minimum spanning tree of $S \cup \{x\} - \{a_i\}$ is better than that of S, x is chosen as a candidate. Find all candidates x for a_i. Randomly update a_i by one of the candidates, say, $a_i = x_0$. Then, a new tree is

generated. Recursively improve the tree and all a_i until convergence

(3.3) **One-to-multiple improving** First of all, we have a fixed integer n_0. If the minimum spanning tree of $S \cup \{x\} - \{a_i\}$ is at least as good as that of S, x is chosen as a candidate. We have two cases:

(i) If the total number of the trees is less than n_0, respectively update a_i by k candidates, i.e. $a_i = x_j$ for $j = 1, 2, \cdots, k$ (k new trees are generated from a_i).

(ii) If the total number of the trees reaches n_0, update a_i by one candidate (e.g. $a_i = x_1$, or a randomly chosen x_j) as in (3.1) or (3.2).

Recursively improve all trees and all a_i until convergence. Finally, an optimal tree is approximated by minimizing all generated trees. The algorithm terminates.

Under the one-to-one improving strategy (3.1) and (3.2), the perturbing-improving algorithm reduces to neighbor-perturbing algorithm (Wu and Gu 2003): Each a_i is updated by only one new x in the neighborhood $N_p(a_i)$ that is better than a_i, i.e. $x \in N_p(a_i)$ and $d^*(x, G) < d^*(a_i, G)$.

Under the one-to-multiple improving strategy (3.3), unless the number of trees reaches n_0, each a_i is updated by multiple new x, say $x_1, x_2, \cdots, x_k$, in $N_p(a_i)(p \geq 2)$ that is at least as good as a_i, i.e. $x_j \in N_p(a_i)(p \geq 2)$ and $d^*(x_j, G) \leq d^*(a_i, G)$ for $j = 1, 2, \cdots, k$. The integer n_0 is chosen to assure the convergence of the algorithm. It can be reasonable large.

4.5 Compare with MGR algorithm

In this section, we compare the perturbing-improving algorithm with the MGR algorithm and show that the perturbing-improving algorithm is $2-$approximation and more efficient than the MGR algorithm.

4.5.1 Accuracy and cost

The perturbing-improving algorithm generates a more accurate tree than the MGR greedy-split algorithm. This is because the perturbing-improving algorithm initially chooses a minimum spanning tree to connect the given genomes. After repeatedly improving, the algorithm obtains a tree better than the minimum spanning tree. The length of the reconstructed tree is within twice of the optimal one (Wu and Gu 2003). It follows that the perturbing-improving algorithm is a $2-$approximation algorithm. However, the MGR algorithm "greedily" connects the given genomes one at a time (Bourque and Pevzner 2002). The reconstructed tree may be far from the optimal tree in the worst case. On the other hand, the perturbing-improving algorithm has a lower computational cost, i.e. $O(m^2n^4)$, than the MGR greedy-split algorithm, i.e. at least $O(m^3n^7)$.

Therefore, the perturbing-improving algorithm is more efficient than the MGR greedy-split algorithm.

4.5.2 Simulation

The perturbing-improving algorithm is implemented into a program *Mgenome*. It is shown by simulations that the perturbing-improving algorithm is more efficient.

At first, a romdan optimal tree T_{opt} is constructed with an optimal ancestral genome s_{opt}. Denote $g_i(i = 1, 2, 3)$ the three genomes of the leaf nodes with $d(s_{opt}, g_i) = d_i(i = 1, 2, 3)$ (see Fig.4.1a).

Next, an ancestral genome s_{app} is reconstructed from g_1, g_2, g_3. The corresponding tree is denoted by T_{app} (see Fig.4.1b).

Then the lengths of the two trees, i.e. $l(T_{app})$ and $l(T_{opt})$, are compared. The efficiency of the algorithm is measured by the ratio $\frac{l(T_{app})}{l(T_{opt})}$, which mainly deponds on d_i.

The above process can be recursively applied to generate a random optimal tree T_{opt} with m genomes. The simulations show that the ratio $\frac{l(T_{app})}{l(T_{opt})} = 1$ (or very close to 1)

for small d_i, i.e. the reconstructed tree is optimal (or very close to optimal). The ratio increases with respective to d_i. For *Mgenome*, $1 \leq \frac{l(T_{app})}{l(T_{opt})} \leq 2$ always holds. However, for MGR, it may have $\frac{l(T_{app})}{l(T_{opt})} > 2$. See Fig.4.1c.

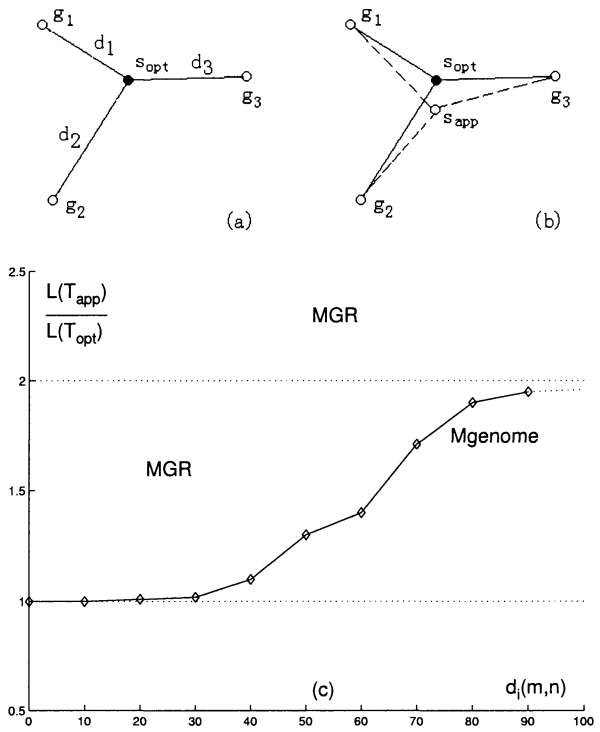

Figure 4.1 (a) From the optimal ancestral genome s_{opt}, an optimal tree T_{opt} is constructed with $d(s_{opt}, g_i) = d_i (i = 1, 2, 3)$. (b) T_{app} is reconstructed from g_1, g_2, g_3 with s_{app} as ancestral genome. (c) $\frac{l(T_{app})}{l(T_{opt})}$ increases with d_i. $1 \leq \frac{l(T_{app})}{l(T_{opt})} \leq 2$ for *Mgenome*. However, MGR may have $\frac{l(T_{app})}{l(T_{opt})} > 2$.

4.5.3 Biological data

We show that *Mgenome* obtains better results than *MGR* for the biological example discussed by Bourque and Pevzner (2002).

Example For the following genomes $G = \{A, B, \cdots, K\}$ (Bourque and Pevzner 2002; Wu and Gu 2003):

(A) Human

1 -32 17 2 23 12 3 20 6 30 7 8 21 31 24 9 -10 -18 11 33 -28 19 14 34 13 25 4 22 -29 26 5 35 -15 -27 -16 -36

(B) Asterina pectinifera

1 30 7 2 23 12 3 -32 6 8 21 31 9 -10 11 19 14 18 33 -13 -5 -22 -4 -25 -20 -36 17 -26 34 -16 -35 15 -24 -27 29 -28

(C) Paracentrotus lividus

1 30 7 2 23 12 3 -32 6 8 21 31 9 -10 11 19 14 18 33 28 -29 27 24 -15 35 16 -34 26 -17 36 20 25 4 22 5 13

(D) Drosophila yakuba

1 25 2 23 17 12 3 20 6 15 30 27 31 18 -19 -9 -21 -8 -7 33 -28 10 11 32 -4 -24 -13 -34 -14 22 -29 26 5 35 -16 -36

(E) Artemia franciscana

1 25 2 23 17 12 3 20 6 15 30 27 31 18 -19 -9 -21 -8 -7 33 -28 10 11 32 -4 -24 -13 -34 -14 26 5 35 -22 -29 -16 -36

(F) Albinaria coerulea

1 34 13 24 28 15 10 9 4 7 11 17 16 19 2 36 35 20 21 -29 -25 -27 -12 -30 -18 -14 -26 -6 -32 31 8 -33 -3 22 5 23

(G) Cepaea nemoralis

1 34 13 24 15 10 28 9 4 7 11 17 16 19 2 36 35 20 21 -29 -25 -27 -12 -30 -18 -14 26 -6 -32 -33 -3 31 8 22 5 23

(H) Katharina tunicata

1 17 2 12 -19 -9 -21 -8 -7 33 -32 -11 -10 28 -4 -25 -24 -13 -34 -14 -26 -16 -36 -35 -29 -20 -18 3 23 15 30 27 22 6 31 5

(I) Lumbricus terrestris

1 27 2 17 36 20 3 29 10 11 35 12 30 21 9 19 18 28 33 7 8 16 26 14 34 13 24 15

32 25 4 22 23 6 31 5

(J) Ascaris suum

1 16 26 17 20 2 21 13 6 9 15 28 34 10 7 35 18 14 32 27 36 4 12 23 25

31 5 22 30 29 19 11 24 3 33 8

(K) Onchocerca volvulus

1 35 10 30 29 11 24 3 23 15 25 27 26 7 14 36 4 19 12 22 20 2 21 13 6

16 32 28 17 34 9 18 31 5 33 8

The *Mgenome* at first finds the pairwise distances of the given genomes. Then the minimum spanning tree of the genomes is taken as the initial Steiner tree. After a series of iterations, the optimal Steiner tree is generated by the *Mgenome*. The total length of the optimal Steiner tree is 149 (Wu and Gu 2003), which is better than 150 that obtained by the **MGR** (Bourque and Pevzner 2002). The optimal ancestral genomes a, d, e, f, h, i and k are reconstructed by the *Mgenome*. See Fig. 4.2.

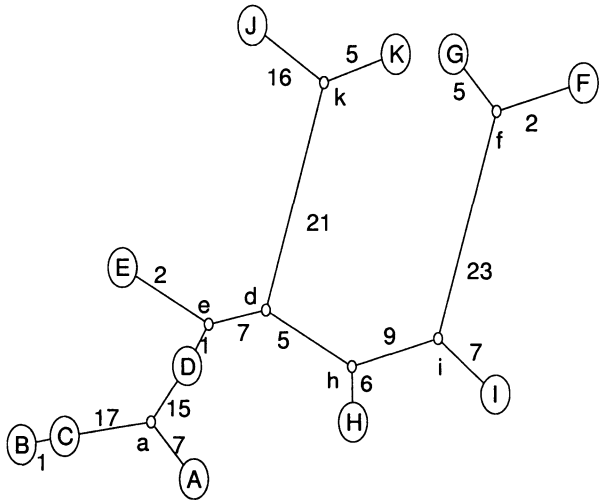

Figure 4.2 The optimal Steiner tree of the 11 genomes.

(a) ancestral genome

1 -32 17 -11 10 -9 -27 18 33 -28 19 14 34 13 25 4 22 -29 26 5 35 30 7 8 21 31 24 2 23 12 3 20 6 -15 -16 -36

(d) ancestral genome

1 25 31 5 -35 -29 -22 -12 -17 -23 -3 14 34 13 24 15 30 27 2 20 6 4 -32 -11 -10 28 -33 7 8 21 9 19 -18 -26 -16 -36

(e) ancestral genome

1 25 2 23 17 12 3 20 6 15 30 27 31 18 -19 -9 -21 -8 -7 33 -28 10 11 32 -4 -24 -13 -34 -14 22 29 26 5 35 -16 -36

(f) ancestral genome

1 34 13 24 15 10 9 4 7 11 17 16 19 2 36 35 20 21 -29 -25 -27 -12 -30 -18 -14 -26 -6 -32 31 8 -33 -28 -3 22 5 23

(h) ancestral genome

1 17 12 -19 -9 -21 -8 -7 33 -28 10 11 32 -4 -25 -23 -3 18 22 29 35 36 16 26 14 34 13 24 15 30 27 2 20 6 31 5

(i) ancestral genome

1 27 2 16 26 14 34 13 24 15 10 11 35 12 -18 -19 -9 -21 -8 -7 -33 -28 30 17 36 20 -4 -25 -23 -3 22 29 32 6 31 5

(k) ancestral genome

1 35 18 27 32 -9 -34 -17 -28 10 30 29 11 24 3 23 -7 -26 -16 -6 -13 -21 -2 -20 -22 -12 -19 -4 -36 -14 15 25 31 5 33 8

4.6 Discussion

We have developed a perturbing-improving algorithm to search for the optimal ancestral genomes for a set of given genomes. The algorithm at first connects all given genomes by a tree with no internal node. Then internal nodes are created and assigned by initial ancestral genomes. And finally, the ancestral genomes are recursively improved until convergence or a fixed number of iteration steps. It is a $2-$approximation algorithm. We show that the algorithm is more efficient than MGR algorithm. The algorithm has potential applications for a wide range of genomes including both and monochromosome multichromosome genomes.

Our discussions are based on signed reversals. However, we can extend the discussions to other mutations, such as insertion, deletion, substitution, reversal, transposition, translocation, fusion, fission, etc.

References

[1] Bader, D. A., Moret, B. M. E. and Yan, M. (2001). *A linear-time algorithm for computing inversion distances between signed permutations with an experimental study.* Lecture Notes in Computer Science, 2125: 365-376.

[2] Bafna, V. and Pevzner, P. (1994). *Genome rearrangements and sorting by reversals.* In Proceeding of 34th IEEE Symposium of the Foundations of Computer Science, 148-157. IEEE Computer Society Press, California.

[3] Bafna, V. and Pevzner, P. (1995). *Sorting permutations by transpositions.* Proceedings of the 6th Annual Symposium on Discrete Algorithms, pages 614-623. ACM Press, January 1995.

[4] Bafna, V. and Pevzner, P. (1996). *Genome rearrangements and sorting by reversals.* SIAM Journal on Computing, 25(2):272-289.

[5] Berman, P. and Hannenhalli, S. (1996). *Fast sorting by reversal.* Proceeding of Combinatorial Pattern Matching (CPM), 168-175. Also Lecture Notes in Computer Science.1075.

[6] Blanchette, M., Bourque, G., and Sankoff, D. (1997). *Breakpoint phylogenies.* In *Genome Information Workshop (GIW 1997),* (eds. Mivano S. and Takagi, T.), pp.25-34. University Academy Press, Tokyo.

[7] Bourque, G. and Pevzner, P. (2002). *Genome-Scale Evolution: Reconstructing Gene Orders in the Ancestral Species.* Genome Research 12:26-36.

[8] Caprara, A. (1997). *Sorting by Reversals is Difficult.* Proceedings of the First Annual International Conference on Computational Molecular Biology (RECOMB'97), ACM Press, New York.

[9] Caprara, A. (1999). *Formulations and hardness of multiple sorting by reversals.* Proceedings of the Third Annual International Conference on Computational Molecular Biology (RECOMB'99), ACM Press, New York.

[10] Cosner, M. E., Jansen, R. K., Moret, B. M. E., Raubeson, L. A., Wang, L. S., Warnow, T., and Wyman, S. (2000). *An empirical comparison of phylogenetic methods on chloroplast gene order data in Campanulaceae.* In Comparative Genomics (DCAF-2000), 104-115, Montreal.

[11] Durbin, R., Eddy, S. R., Krogh, A. and Mitchison, G. (1998). *Biological sequence analysis: Probabilistic models of proteins and nucleic acids.* Cambridge University Press, Cambridge, UK.

[12] Felsenstein, J. (1973). *Maximum likelihood and minimum-steps methods for estimating evolutionary trees from data on discrete characters.* Systematic Zoology, 22:240-249.

[13] Gu, X., (1999). *Statistical methods for testing functional divergence after gene duplication.* Molecular Biology and Evolution 16:1664-1674.

[14] Gu, X. (2000). *A simple evolutionary model for genome phylogeny inference based on gene content.* Comparative genomics (ed. Sankoff and Nadeau), pp.515-524. Kluwer Academic Publishers, The Netherlands .

[15] Gu, X. (2001). *Maximum likelihood approach for gene family evolution under functional divergence.* Molecular Biology and Evolution. 18: 453-464.

[16] Gusfield, D. (1997). *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, Cambridge, UK.

[17] Graur, D. and Li, W. (1997). *Fundamentals of Molecular Evolution.* Sinauer Association, Inc. Massachusetts USA.

[18] Gu, X. and Zhang, Z. (2004). *Genome Phylogenetic Analysis Based on Extended Gene Contents. Molecular Biology and Evolution*, 21(7):1401-1408.

[19] Hannenhalli, S. and Pevzner, P. (1995a). *Transforming men into mice (polynomial algorithm for genomic distance problems.* Proceeding of IEEE Symposium of the Foundations of Computer Science.

[20] Hannenhalli, S. and Pevzner, P. (1995b). *Transforming cabbage into turnip (polynomial algorithm for sorting signed permutations by reversals).* Proceeding of of 27th Annual ACM Symposium on Theory of Computing, 178-189.

[21] Nei, M. and S. Kumar., S. (2000). *Molecular Evolution and Phylogenetics.* Oxford University Press, New York.

[22] Olmstead, R. G. and Sweere, J. A. (1994). *Combining data in phylogenetic systematics: an empirical approach using three molecular data sets in the Solanaceae.* Systems Biolology. 43:467-481.

[23] Palmer, A. R. (1992). *Calcification in marine molluscs: How costly is it?* Proceedings of the National Academy of Sciences, 89:1379-1382.

[24] Palmer, D. and Herbon, L. A. (1988). *Plant mitochondrial dna evolves rapidly in structure, but slowly in sequence.* Journal of Molecular Evolution, 27:87-97.

[25] Sankoff, D. (1999). *Genome rearrangement with gene families.* Bioinfomatics, 15:909-917.

[26] Sankoff, D. and Blanchette, M. (1998). *Multiple genome rearrangement and breakpoint phologeny.* Journal of Computational Biology, 5: 555-570.

[27] Sankoff, D., Sudaram, G. and Kececioglu, J. (1996). *Steiner points in the space of genome rearrangements.* International Journal of the Foundations of Computer Science, 7:1-9.

[28] Watterson, G. A. and Ewens, W. J. and Hall, T. E. and Morgan, A. (1982). *The chromosome inversion problem.* Journal of Theoretical Biology, 99, 1-7.

[29] Wu, S. and Gu, X. (2002). *Multiple Genome Rearrangement By Reversals.* Pacific Symposium on Biocomputing 7:259-270.

[30] Wu, S. and Gu, X. (2003). *Algorithms for Multiple Genome Rearrangement by Signed Reversals.* Pacific Symposium on Biocomputing 8:363-374.

CHAPTER 5. A partitioning algorithm for large scale multiple genome rearrangement by signed reversals

(A paper to be submitted)

Shiquan Wu and Xun Gu

Abstract

For a small collection of genomes, the evolutionary history of genome rearrangement by signed reversals can be efficiently reconstructed by various algorithms, including MGR greedy-split, neighbor-perturbing, perturbing-improving algorithm. However, for a large collection of genomes, it may be hard to reconstruct an accurate phylogenetic tree in the least computational cost. All existing algorithms are approximations, their computational costs are usually high and their accuracies of phylogenetic trees may be decreased with genome scales. In this paper, we develop a partitioning algorithm to reconstruct the evolutionary history for a large scale multiple genome rearrangement by signed reversals. The algorithm partitions a large collection of genomes into a series of small groups whose evolutionary histories can be efficiently reconstructed by existing algorithms. The evolutionary history of the large collection of genomes is approximated by joining all evolutionary histories of the small groups.

5.1 Introduction

To reconstruct evolutionary history under edit distance defined by various types of mutations is a challenging problem in bioinformatics. Both inference accuracy and computational cost are two major issues. When only local mutations (insertion, deletion, and substitution) are considered, the evolutionary history is the traditional phylogenetic tree and can be efficiently inferred by different methods such as the sequence alignment method [3, 6, 7, 9, 11], the parsimony method [3, 5, 6], the maximum likelihood method [3, 4, 6], the Kolmogorov complexity method [8, 12, 13, 14], and so on. However, if non-local mutations (such as reversal, transposition, fission, fusion, etc.) are allowed, evolutionary history inference becomes a multiple genome rearrangement problem and is NP-hard. Various approximation algorithms are designed to solve the problem, including the grid search algorithm [10], the nearest path search algorithm [15], the MGR greedy split algorithm [2], the branch-and-bound algorithm [16], the neighbor-perturbing algorithm [16] and the perturbing-improving algorithm [17]. These approximation algorithms can efficiently reconstruct the evolutionary history for small scale of genomes. The inference accuracy is mainly affected by the number of genomes. The computational cost depends not only on the number of genomes, but also the number of genes contained in genomes.

For a large scale of genomes (which may contain a large number of genomes and each genome may contain a large number of genes), most algorithms, e.g. the grid search algorithm and the MGR greedy split algorithm, reconstruct an approximated phylogenetic tree by recursively joining genomes one at a time. Therefore, the computational cost is usually high. For example, the running time of the *MGR* algorithm increases rapidly with respective to the genome size. On the other hand, since the existing algorithms are approximations, the accuracy of the reconstructed phylogenetic tree may also be low for a large collection of genomes. For example, the *MGR* algorithm may generate a tree far

from the optimal tree. In practical biological applications, it is essential to reconstruct accurate phylogenetic trees for a large scale of genomes in the least cost. In this paper, we develop a partitioning algorithm to reconstruct the evolutionary history for a large scale multiple genome rearrangement by signed reversals. The algorithm partitions a large collection of genomes into a series of small groups so that their evolutionary histories can be efficiently reconstructed by existing algorithms. The evolutionary history of the large collection of genomes is approximated by joining together all evolutionary histories of the small groups by minimizing the total distance.

5.2 Previous efficient algorithms

For small scale genomes, the evolutionary history can be efficiently reconstructed by many approximation algorithms such as MGR greedy-split algorithm, neighbor-perturbing algorithm, perturbing-improving algorithm, and so on. Here we have an overview of these algorithms.

MGR Greedy-split algorithm [2] recursively reconstructs the evolutionary history of genomes by greedy split. Suppose we are given a set of genomes: $g_1, g_2, \cdots, g_m$. The algorithm initially joins the closest three genomes: g_1, g_2 and g_3. If $g_1, g_2, \cdots, g_{m-1}$ are joined in a tree. The algorithm searches for a split edge with a split site among the edges of the tree. Then g_m is joined to the split site (see Fig.3.1b on Page 42).

Neighbor-perturbing algorithm [16] searches for the evolutionary history of genomes by perturbing an arbitrary Steiner tree neighborhood by neighborhood and improving it until convergence (see Fig.3.2 on Page 46). It is a 2-approximation algorithm.

Perturbing-improving algorithm [17] searches for the optimal ancestral genomes of

the optimal phylogenetic tree of given genomes. It improves the neighbor-perturbing algorithm by updating each ancestral genome with multiple new ones that are at least as good as the old one. This algorithm is much more efficient than MGR algorithm.

These algorithms are efficient for small scale of genomes. However, when the genome scale is getting larger, the computatinal cost becomes more expensive. In addition, the accuracy of the reconstructed phylogenetic tree is getting lower.

5.3 Partitioning algorithm

Usually, similar genomes are found closer together in the phylogenetic tree, while more distant genomes usually appear farther apart in the tree. The tree can be partitioned into parts. Within each part, the genomes are more similar and are locally connected by the tree. Based on the observation, a large scale of genomes can be partitioned into a series of small scale of genomes. We can at first find the trees for all small scale genomes and then join these trees together to form a tree for the original large scale genomes. Through reconstructing trees block by block, we may decrease the possibility joining nodes far from each other. This may increase the accuracy of reconstructing trees. On the other hand, we may decrease the possibility processing unrelated nodes together and save the computational costs.

The partitioning algorithm consists of three steps. The first step is genome partitioning, which partitions a large collection of genomes into a series of small groups. The second step is tree reconstructing, which respectively reconstructs the phylogenetic trees for the small groups. The third step is tree connecting, which connects all trees of the small groups together to form a tree for the original large scale genomes.

Step 1: Genome partitioning

In the genome partitioning step, the algorithm may partition the given collection of genomes in two ways: Bisecting and nearest-grouping.

(1) **Bisecting** For a given large collection of genomes $G = \{g_j | j = 1, 2, \cdots, m\}$, the partitioning algorithm partitions G into a series of groups of genomes $G_i = \{g_{j_i} | i = 1, 2, \cdots, m_i\}$ such that (1) $\sum_{i \geq 1} m_i = m$ and (2) each $m_i \leq m_0$, where m_0 is a fixed integer such that the evolutionary history of m_0 genomes can be efficiently reconstructed by existing algorithms. The maximum group size m_0 depends on the algorithms we choose. For neighbor-perturbing [16] or perturbing-improving algorithm [17], we may choose $m_0 \leq 30$.

The partitioning process is as follows. First of all, find the maximum pairwise signed reversal distance, say $d(g_{j_1}, g_{j_2})$. Denote $G_1 = \{g_{j_1}\}$ and $G_2 = \{g_{j_2}\}$. For any $g_j \in G$, if $d(g_j, g_{j_1}) < d(g_j, g_{j_2})$, put g_j into G_1. Otherwise, put g_j into G_2. G is then partitioned into G_1 and G_2. If G_1 (or G_2) contains more than m_0 genomes, repeat the above partitioning process for G_1 (or G_2). Recursively, G is partitioned into a series of $G_i (i = 1, 2, \cdots, m_i)$ such that $\sum_{i \geq 1} m_i = m$ and each $m_i \leq m_0$.

(2) **Nearest-grouping** Initially, each single genome forms a group, i.e. $G_i = \{g_i\}$ for $i = 1, 2, \cdots, m$. Next, find the closest pair, say G_1 and G_2, such that $d(G_1, G_2) \leq d(G_i, G_j) = min\{d(x, y) | x \in G_i, y \in G_j\}$ for all i and j. If G_1 and G_2 have less than m_0 genomes, unite G_1 and G_2 to form a new group. Repeat the union process for all groups until convergence. We then get $G_i (i = 1, 2, \cdots, m_i)$ such that $\sum_{i \geq 1} m_i = m$ $(m_i \leq m_0)$.

Step 2: Tree reconstructing

In this tree reconstructing step, neighbor-perturbing, or perturbing-improving algorithm is applied to reconstructing the evolutionary history T_i of G_i. Because each G_i contains m_i genomes and $m_i \leq m_0$, T_i can be efficiently reconstructed.

Step 3: Tree connecting

In the tree connecting step, the pairwise distance d_{ij} between each pair T_i and T_j is computed: $d_{ij} = d(T_i, T_j) = min\{d(x, y)|x \in T_i, y \in T_j\}$. Each d_{ij} corresponds to a node u_{ij} in the paths of T_i and another node v_{ij} in the paths of T_j.

Finally, all T_i are connected by minimizing the total distance. Choose the minimum d_{ij}, join $u_{ij} \in T_i$ with $v_{ij} \in T_j$ if the joining still forms a tree. Repeat the joining process for all tree T_i, a tree is obtained. Furthermore, improve the tree by perturbing u_{ij} and v_{ij}, respectively.

We now describe the algorithm as follows.

Algorithm **Partitioning**

Input Genomes $G = \{g_1, g_2, \cdots, g_m\}$.

Output Optimal phylogenetic tree.

Step 1 Genome partitioning:

By bisecting:(1.1b)+(1.2b) or nearest-grouping: (1.1n)+(1.2n)

Bisecting

(1.1b) $G_1 = G_2 = \emptyset$.

Find the maximum signed reversal distance $d(g_{j_1}, g_{j_2})$.

For any $g_j \in G$, if $d(g_j, g_{j_1}) < d(g_j, g_{j_2})$, $G_1 = G_1 \cup \{g_j\}$.

Otherwise, $G_2 = G_2 \cup \{g_j\}$. See Fig.5.1(a).

(1.2b) Replace G by G_1 and G_2, respectively. Repeat (1.1b) until each G_i contains at most m_0 genomes. See Fig.5.1(b)(c).

Nearest-grouping

(1.1n) $G_i = \{g_i\}$ for $i = 1, 2, \cdots, m$.

(1.2n) Find the closest pair G_1 and G_2: $|G_1 \cup G_2| \leq m_0$ and

$d(G_1, G_2) \leq d(G_i, G_j) = min\{d(x, y) | x \in G_i, y \in G_j\}$.

Redefine $G_1 = G_2 = G_1 \cup G_2$.

Repeat this step until convergence.

(1.3) Finally, $G = \cup_{i \geq 1} G_i$ with $|G_i| \leq m_0$. See Fig.5.1(c).

Step 2 Tree reconstructing:

(2.1) For each G_i, the evolutionary history T_i of G_i is reconstructed by neighbor-perturbing/perturbing-improving algorithm.

(2.2) A collection $\{T_i | i = 1, 2, \cdots\}$ is obtained. See Fig.5.1(d).

Step 3 Tree connecting:

(3.1) Compute the distance d_{ij} for each pair T_i and T_j.

Denote $u_{ij} \in T_i$ and $v_{ij} \in T_j$ with $d_{ij} = d(u_{ij}, v_{ij})$.

(3.2) Connect all $\{T_i | i = 1, 2, \cdots\}$ to form a tree by minimizing the total distance:

(i) Choose the minimum d_{ij}.

(ii) Join u_{ij} and v_{ij} by a shortest reversal path if the joining still forms a tree.

(iii) Repeat the joining process until convergence.

(3.3) Improve the tree by perturbing u_{ij} and v_{ij}, respectively.

See Fig.5.1(e)(f).

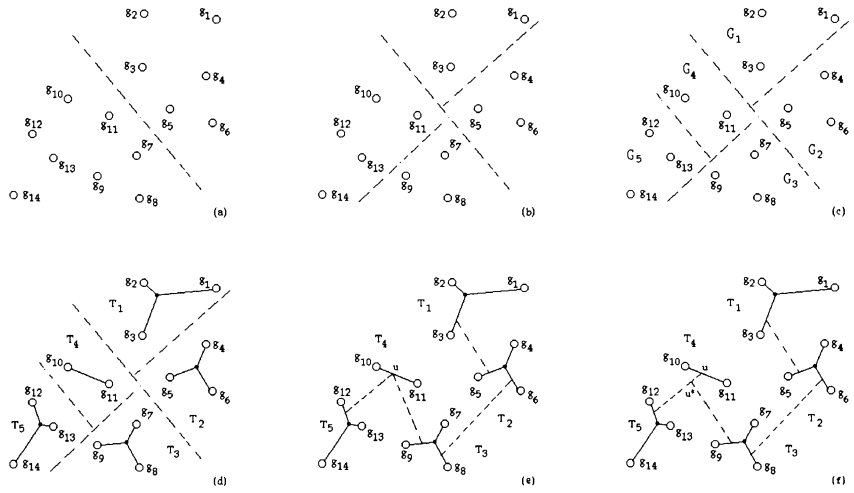

Figure 5.1 Partition process: (a) Genomes $g_i(1 \leq i \leq 14)$ are partitioned into two parts: $g_i(1 \leq i \leq 6)$ and $g_i(7 \leq i \leq 14)$. (b) $g_i(1 \leq i \leq 6)$ are partitioned into two parts: $g_i(1 \leq i \leq 3)$ and $g_i(4 \leq i \leq 6)$. Meanwhile, $g_i(7 \leq i \leq 14)$ are partitioned into another two parts: $g_i(7 \leq i \leq 9)$ and $g_i(10 \leq i \leq 14)$. (c) $g_i(10 \leq i \leq 14)$ are partitioned into two parts: $g_i(i = 10, 11)$ and $g_i(12 \leq i \leq 14)$. Therefore, $g_i(1 \leq i \leq 14)$ are partitioned into $G_i(1 \leq i \leq 5)$, each of which contains at most $m_0(= 3)$ genomes. (d) A tree T_i is reconstructed for G_i $(1 \leq i \leq 5)$. (e) All T_i $(1 \leq i \leq 5)$ are connected to form a phylogenetic tree for $g_i(1 \leq i \leq 14)$. (f) u is improved into u^* by a series of perturbing.

5.4 Discussion

It is a great challenge to reconstruct the evolutionary history of a large number of genomes. The partitioning algorithm is designed to partition a large number of genomes into a series of small groups of genomes whose evolutionary histories can be efficiently reconstructed by existing algorithms. The evolutionary history of the whole set of genomes is approximated by connecting the evolutionary histories of all subgroups. The accuracy of the evolutionary history of the large scale genomes depends on the sizes

m_0 of subgroups. In practical applications, one should try various m_0 and choose the best tree from all possible trees.

The partition in this paper is done by bisecting and nearest-grouping. It can also be done in any other ways as long as the partition can give rise to a more accurate tree and a lower computational cost.

There are many other methods that can efficiently reconstruct the evolutionary history for a large scale of genomes. For example, a simulated annealing method may be one of such possible ways.

Furthermore, a large scale of genomes may contain a large number of genomes and each genome may contain a large number of genes. In this paper, we only deal with a large number of genomes. If genomes contain a large number of genes, we may partition the genes into groups. For each group of genes, we reconstruct a phylogenetic tree for the genomes. Then we take an optimal reconciled tree for all phylogenetic trees as the phylogenetic tree of the original genomes.

References

[1] Bader, D. A., Moret, B. M. E. and Yan, M. (2001). *A linear-time algorithm for computing inversion distances between signed permutations with an experimental study.* Lecture Notes in Computer Science, 2125: 365-376.

[2] Bourque, G. and Pevzner, P. (2002). *Genome-Scale Evolution: Reconstructing Gene Orders in the Ancestral Species.* Genome Research 12: 26-36.

[3] Durbin, R., Eddy, S. R., Krogh, A. and Mitchison, G. (1998). *Biological sequence analysis: Probabilistic models of proteins and nucleic acids*, Cambridge University Press, Cambridge, UK.

[4] Felsenstein, J. (1981). *Evolutionary trees from DNA sequences: a maximum likelihood approach.* Journal of Molecular Evolution, 17:368-376.

[5] Fitch, W. M. (1971). *Toward defining the course of evolution: minimum change for a specific tree topology.* Systems Zoology, 35:406-416.

[6] Gusfield, D. (1997). *Algorithms on Strings, Trees, and Sequences: Computer Science and Computational Biology.* Cambridge University Press, Cambridge, UK.

[7] Lake, J. A. (1994). *Reconstructing evolutionary trees from DNA and protein sequences: paralinear distances.* Proceeding of National Science,USA, 91:1455-1459.

[8] Otu, H. H. and Sayood, K. (2003). *A new sequence distance measure for phylogenetic tree construction.* Bioinformatics, 19:2122-2130.

[9] Saitou, N. and Nei, M. (1987). *The neighbor-joining method: a new method for reconstructing phylogenetic trees.* Molecular Biology Evolution, 4: 406-425.

[10] Sankoff, D., Sudaram, G. and Kececioglu, J. (1996). *Steiner points in the space of genome rearrangements.* International Journal of the Foundations of Computer Science, 7:1-9.

[11] Sneath, P. H. A. and Sokal, R. R. (1973). *Numerical Taxonomy*, pp.230-234, W.H. Freeman and Company, San Francisco, USA.

[12] Snel, B., Bork, P. and Huynen, M. A. (1999). *Genome phylogeny based on gene content.* Nature Genetics, 21: 108-110.

[13] Snel, B., Bork, P. and Huynen, M. A. (2000). *Genome evolution: gene fusion versus gene fission.* Trends Genetics, 16: 9-11.

[14] Snel, B., Bork, P. and Huynen, M. A. (2002). *Genomes in flux: the evolution of archaeal and proteobacterial gene content.* Genome Research, 12:17-25.

[15] Wu, S. and Gu, X. (2002). *Multiple Genome Rearrangement By Reversals.* Pacific Symposium on Biocomputing 7:259-270.

[16] Wu, S. and Gu, X. (2003). *Algorithms for Multiple Genome Rearrangement by Signed Reversals.* Pacific Symposium on Biocomputing 8:363-374.

[17] Wu, S. and Gu, X. (2003). *A perturbing-improving algorithm for reconstructing ancestral genomes*, to be submitted.

CHAPTER 6. General conclusions

6.1 Summary of algorithms

We have discussed multiple genome rearrangement by signed reversal and developed several approximation algorithms: the nearest path search algorithm, the branch-and-bound algorithm, the neighbor-perturbing algorithm, the perturbing-improving algorithm, and the partitioning algorithm. These algorithms are shown to be more efficient than other similar ones on both the accuracy and computational cost by theoretical proofs, computer simulations, and biological examples. Specifically, the nearest path search algorithm is better than Sankoff's grid search algorithm. The neighbor-perturbing algorithm is better than Bourque and Pevzner's MGR greedy split algorithm. Among our algorithms, the neighbor-perturbing algorithm is better than the nearest path search algorithm. The perturbing-improving algorithm is the best one of all algorithms. The partitioning algorithm deals with large scale genomes based on other algorithms.

By applying our algorithms and the programs implemented from the algorithms, we can solve a wide range of multiple genome rearrangement by signed reversal. The algorithms can be applied to deal with problems involved more types of mutations. However, the accuracy and computational cost may be affected.

6.2 Further improvement

Further work can be done on multiple genome rearrangement. This includes improving the model and the algorithms in the following three ways.

(1) More types of mutations

Because only signed reversal is allowed in our multiple genome rearrangement problem, it is still biological unrealistic. During evolution, various types of mutations may occur randomly. In order to solve the problems from real biological applications, we need to extend the algorithms to handle insertion, deletion, substitution, reversal, transposition, translocation, fusion, fission, recombination, gene duplication, gene loss, etc.

(2) More copies of genes

The genomes we discuss always contain only one single copy of each gene. This is a strong restriction. In practical biological applications, genomes may contain multiple copies of genes. So we also need to allow multiple copies of genes for multiple genome rearrangement.

(3) Random search strategy

In our algorithms, we always check all over the neighborhoods of Steiner nodes to improve trees. This gives rise to the expensive computational cost. Random search strategy can be applied in checking the neighborhoods to reduce the computational cost and still keep accuracy.

6.3 General problem

We conclude from our discussions that a more general multiple genome rearrangement problem should be discussed: For a collection of genomes that may contain multi-chromosomes and multiple copies of genes, reconstruct the evolutionary history under a set of mutations such as insertion, deletion, substitution, reversal (signed or unsigned), transposition, translocation, fusion, fission, recombination, gene duplication, gene loss, and so on.

The problem is NP-hard. It is challenging to develop efficient approximation algorithms to solve the problem.

ACKNOWLEDGEMENTS

I am sincerely indebted to my major professor Dr.Xun Gu. It is he who advised me to choose comparative genomics as my research topic. His enthusiasm and integral views on science and research greatly benefit me in my learning and research. With his supervising, I designed algorithms by not only theoretical approaches, but also computer simulations and biological applications. Through these ways, our algorithms are developed more efficient than other similar previous ones.

My special thank also goes to my POS committee members for their strong support for my research. Dr.Stephen Willson suggested me to improve the neighbor-perturbing algorithm by removing strong restrictions. This is one of the important ideas for the perturbing-improving algorithm. By Dr.Zhijun Wu's helpful ideas, I focus all of my discussions in solving one case of the problem and then obtained a series of efficient algorithms. I conducted a lot of simulations to verify the algorithms. This was suggested by Dr.Xun Gu. From Dr.Karin Dorman (and several papers she introduced to me), I also got helpful ideas on simulations and algorithm design (giving rise to the section "Simulations on optimal trees"). Dr.Xiaoqiu Huang gave me very good new ideas about the general model of multiple genome rearrangement. Also with his helpful suggestions, I reduced the running time of our programs by specially dealing with the conserved segments of genomes.